SISTEMI OPERATIVI

(MODULO DI INFORMATICA II)

La memoria virtuale

Prof. Luca Gherardi

Prof.ssa Patrizia Scandurra (anni precedenti)

Università degli Studi di Bergamo

a.a. 2012-13

Sommario

- La memoria virtuale
- Richiesta di paginazione
- Creazione di un processo
  - Copia durante la scrittura
  - File mappati in memoria
- Sostituzione della pagina
- Allocazione dei frame
- Altre considerazioni

Richiami sulla gestione della memoria centrale

- **Allocazione non contigua**:
  - L'immagine di un processo NON è un blocco unico
  - L'immagine di un processo viene spezzata in più parti che sono caricate in memoria in aree non contigue
- Due tecniche fondamentali, spesso combinate:
  - **Paginazione:** l'immagine di un processo è divisa in parti (pagine) di dimensione fissa per un certo SO
    - problemi simili alla gestione a partizioni fisse (frammentazione interna nell'ultima pagina)
  - **Segmentazione:** le parti (segmenti) sono di lunghezza variabile e riflettono la logica del programma (es: dati, istruzioni)
    - problemi simili alla gestione a partizioni variabili (frammentazione esterna)

Introduzione al problema della Memoria virtuale (1)

- Le tecniche di gestione della memoria fisica richiedono che l'**intero processo sia in memoria fisica** (centrale) per poter essere eseguito
  - Un programma non può essere più grande della memoria fisica
- In molti casi però il programma non viene eseguito interamente:
  - Codice per la gestione di condizione di errore poco probabili
  - Array, tabelle e liste sono tipicamente allocati con una dimensione maggiore della necessaria
    - Il costruttore `ArrayList()` crea un ArrayList di dimensione 10. Potrebbero non essere usati tutti
  - Alcune opzioni o funzionalità di un programma sono usate solo di rado (e.g. backup)

Introduzione al problema della Memoria virtuale (2)

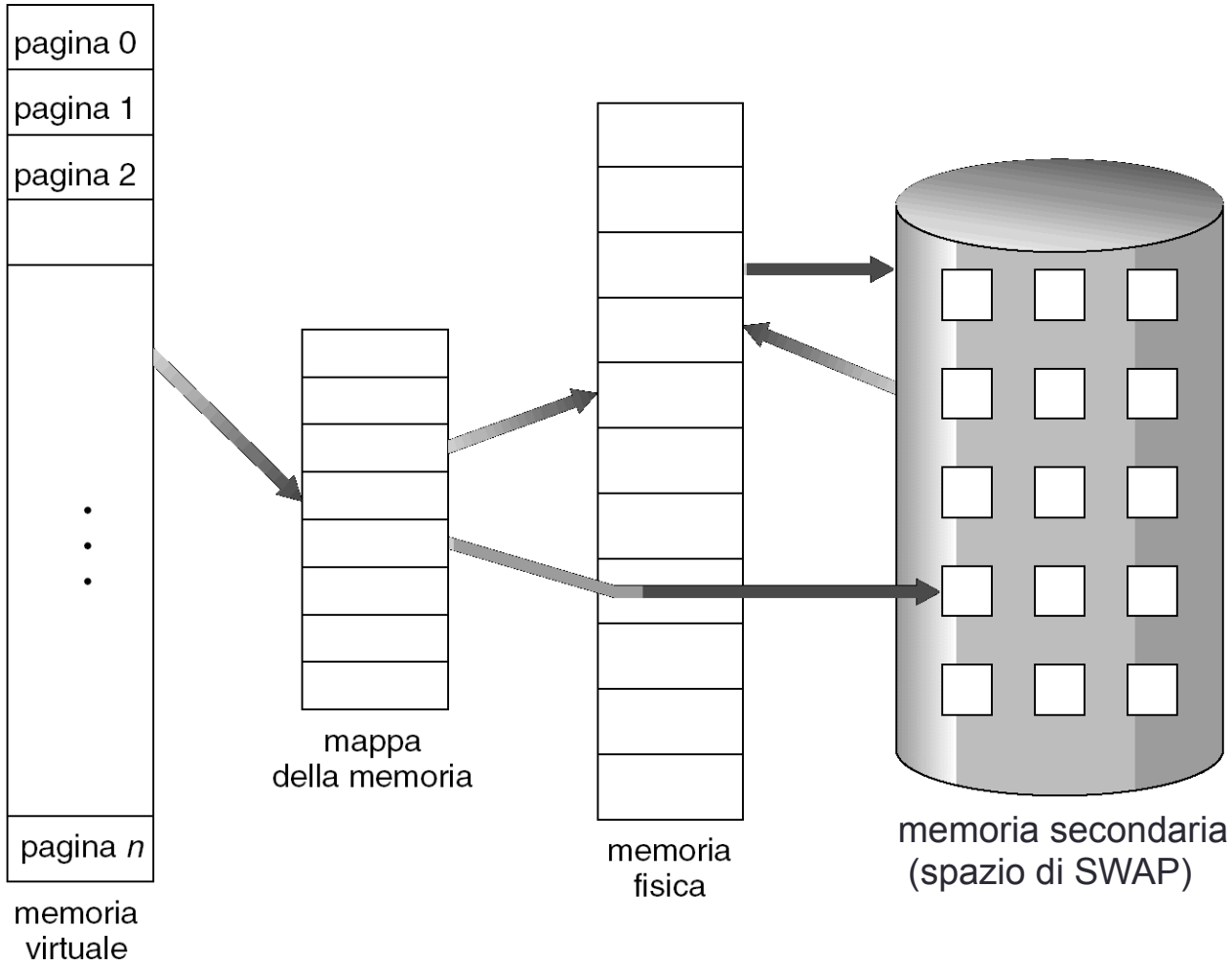
- Sarebbe quindi conveniente poter caricare solo una parte del programma in memoria fisica
- Vantaggi:
  - Un programma può avere dimensioni maggiori di quelle della memoria fisica
  - Molti più programmi possono stare in memoria centrale contemporaneamente
    - Meno tempo necessario per lo swapping, più efficienza della CPU
  - Per caricare un programma in memoria centrale sono necessarie meno operazioni di I/O
    - Programmi lanciati in modo più rapido
- Questo è possibile attraverso la **memoria virtuale**

Memoria virtuale

- L'idea di memoria virtuale si basa sul
  - separare la visione di memoria percepita dall'utente (memoria logica)
  - dalla memoria fisica
- Il termine **virtuale** indica una cosa che **non esiste fisicamente ma solo logicamente** (concettualmente, metaforicamente,…)
- Lo **spazio degli indirizzi virtuali** rappresenta la collocazione dei processi nella memoria virtuale
- La memoria virtuale può essere implementata attraverso:
  - **Paginazione su richiesta**
  - **Domanda di segmentazione** (non in programma)

Schema della memoria virtuale

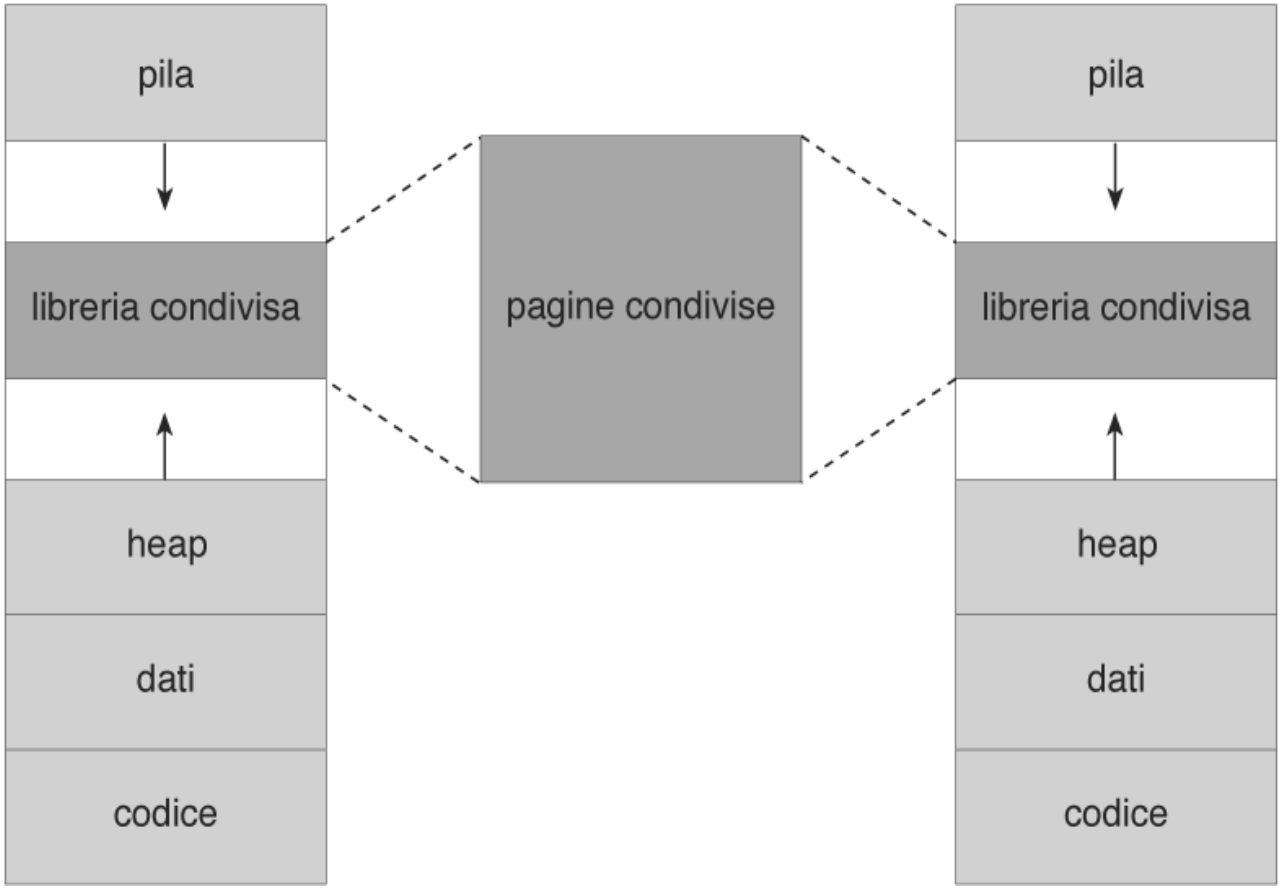
Esiste una mappa di memoria che memorizza la relazione tra una pagina virtuale e una pagina fisica



Altri vantaggi della memoria virtuale

- Oltre ai vantaggi visti prima la memoria virtuale facilità la condivisione di memoria e file tra i processi
  - Le librerie di sistema possono essere condivise mappando le stesse pagine fisiche a diverse pagine virtuali
    - Ogni processo vede la libreria come parte del suo spazio degli indirizzi virtuali
  - Allo stesso modo un processo può condividere parte del suo spazio degli indirizzi con altri processi (comunicazione basata su memoria condivisa)
  - La chiamata fork() può essere eseguita in modo più veloce
    - Le pagine possono essere condivise tra padre e figlio

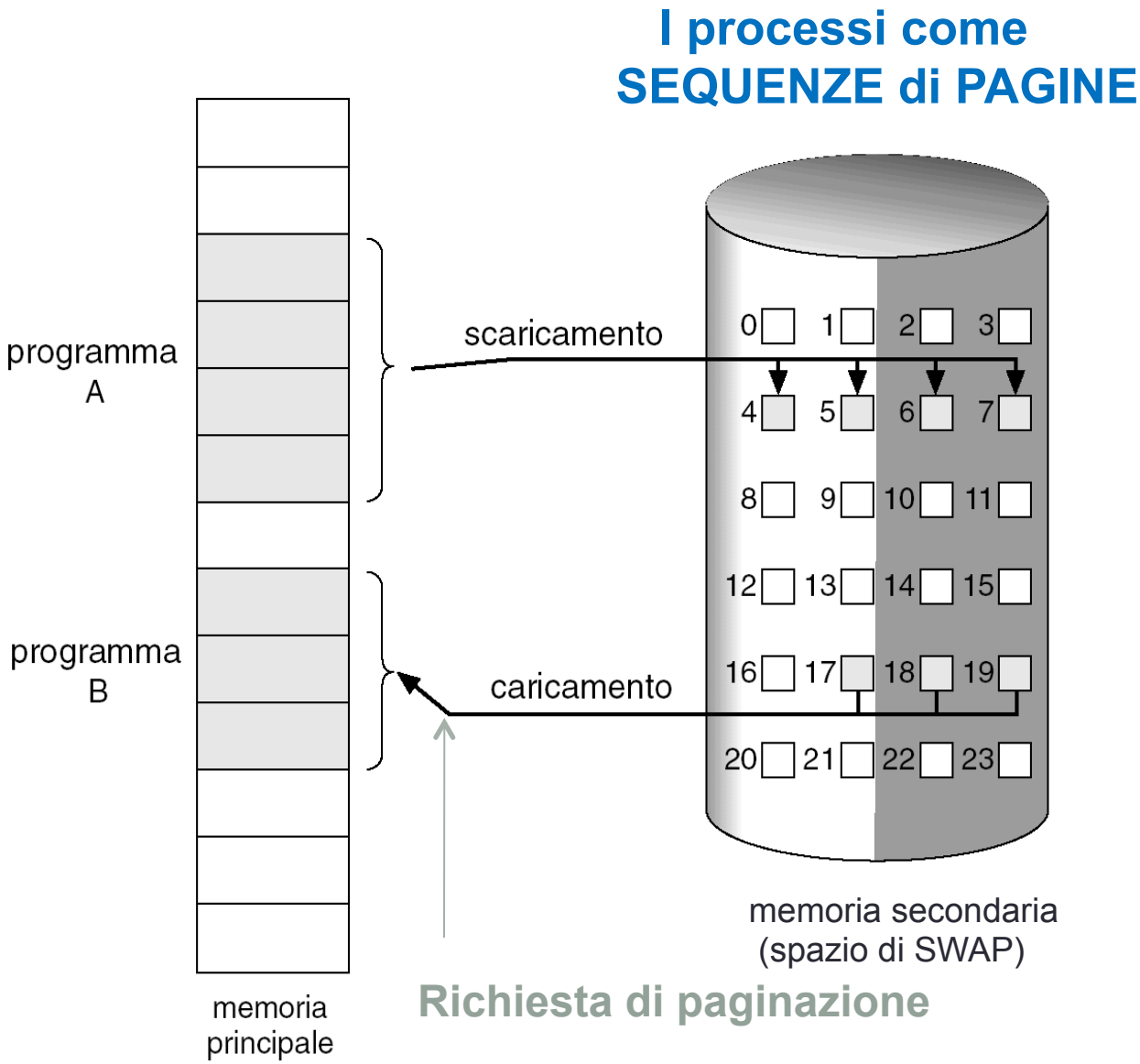
Condivisione delle pagine



Paginazione su richiesta

- Nel momento in cui è necessario caricare in memoria un eseguibile residente su disco
  - Anziché caricare immediatamente tutte le pagine nella moria fisica
  - **Si caricano le pagine solo nel momento in cui realmente servono**
- Le pagine non utilizzate non sono mai caricate in memoria
- È una tecnica analoga all'avvicendamento dei processi (swapping)
  - Ma vale per le pagine anziché per l'intero processo
  - È detta anche lazy swapping
- Il modulo responsabile di caricare le pagine è detto **paginatore o pager**

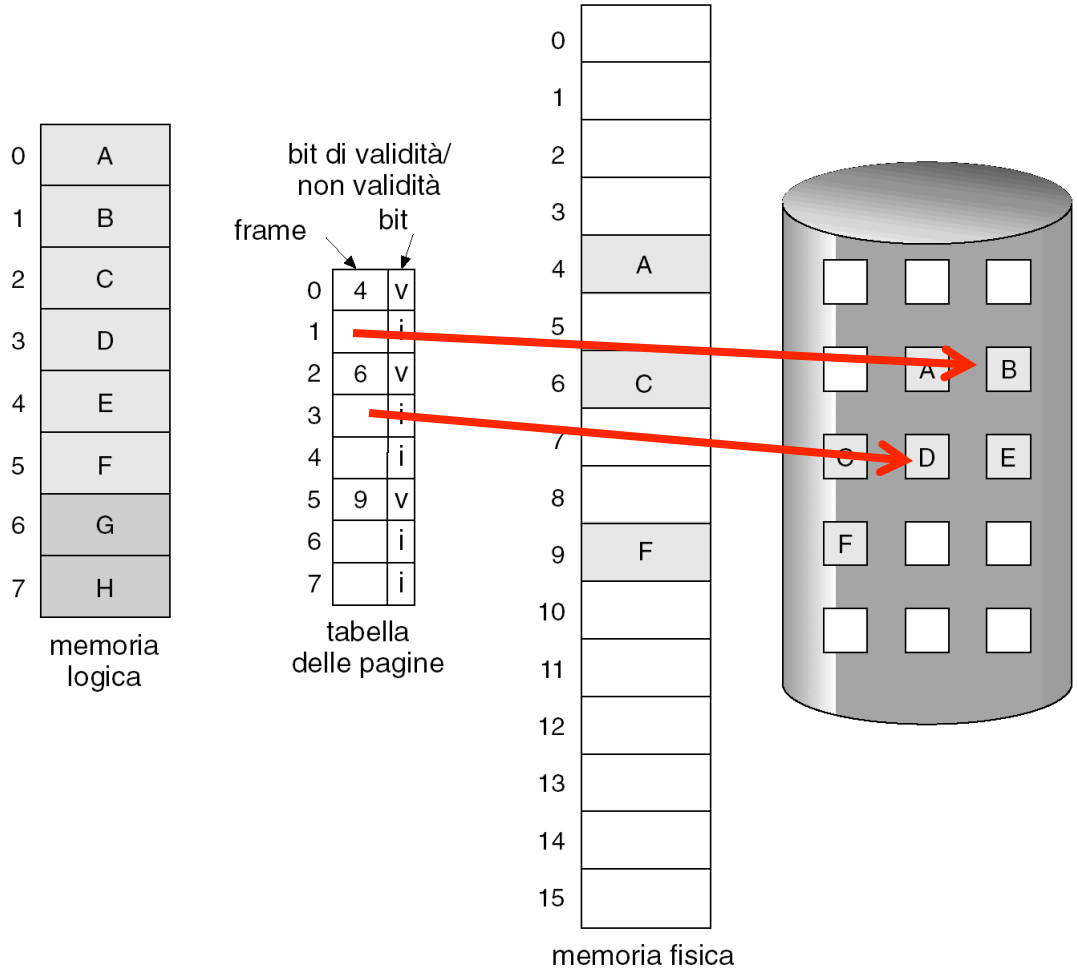
Paginazione su richiesta – schema



Paginazione su richiesta
Concetti fondamentali (1)

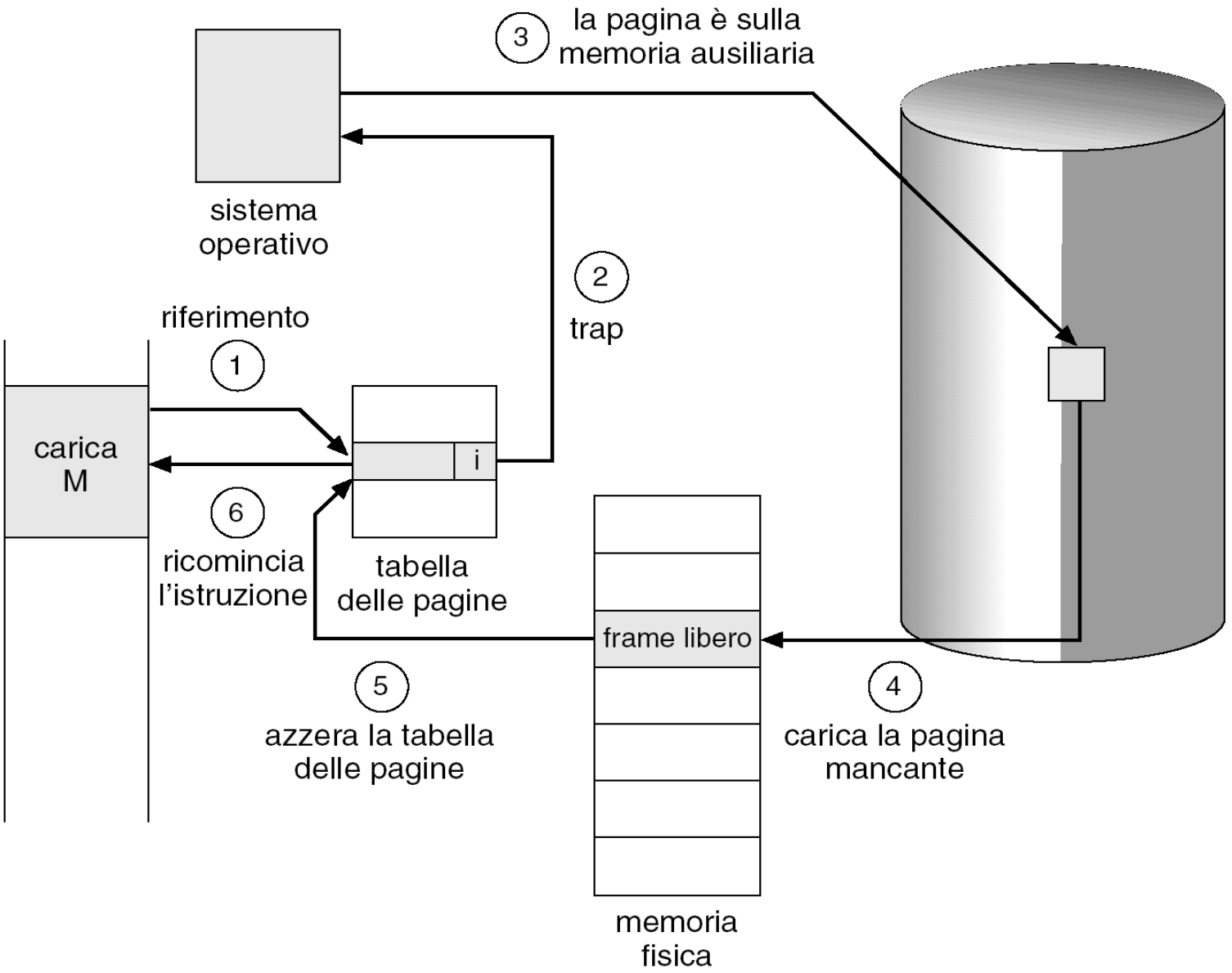
- Inizialmente il paginatore ipotizza quali pagine del processo saranno realmente utilizzate
  - E carica in memoria solo quelle pagine
- Per implementare la richiesta di paginazione è necessario disporre di un'architettura per distinguere tra pagine caricate e non caricate
- Si utilizza lo schema del **bit di validità**
  - **Valido**: pagina appartenente allo spazio degli indirizzi del processo **e** caricata in memoria
  - **Non valido**: pagina non appartenente allo spazio degli indirizzi del processo **o** non caricata in memoria
- L'elemento della tabella della pagine corrispondente a una pagina non valida
  - È contrassegnato come non valido o contiene l'indirizzo della pagina nel disco

Paginazione su richiesta
Concetti fondamentali (1)



Gestire una mancanza di pagina

- Se il processo tenta l'accesso ad una pagina non valida viene sollevata un'**eccezione di pagina mancante** (*page fault*)
- Gestione
  1. Si esamina una tabella interna al processo per decidere:
     - Pagina non valida ➔ il processo viene terminato
     - Pagina non in memoria ➔ si procede
  2. Si cerca un frame libero sufficientemente grande
  3. Si caricare dal disco la pagina nel frame individuato
  4. Si modificano la tabella interna al processo e la validità del bit
  5. Si riavvia l'istruzione interrotta dall'eccezione


3
la pagina è sulla
memoria ausiliaria
sistema
operativo
2
trap
riferimento
1
carica
M
i
6
ricomincia
l'istruzione
tabella
delle pagine
frame libero
5
azzera la tabella
delle pagine
4
carica la pagina
mancante
memoria
fisica

Prestazione della richiesta di paginazione

- Probabilità di mancanza di pagina $0 \leq p \leq 1.0$
  - se $p = 0$ non ci sono mancanze di pagina
  - se $p = 1$, ogni riferimento è una mancanza di pagina

- Tempo di accesso effettivo (EAT)

  EAT = (1 – *p*) x tempo di accesso alla memoria

  + *p* x tempo di mancanza della pagina

dove:

tempo di mancanza della pagina =

tempo per attivare il servizio di page fault + swap page in + tempo di ripresa del processo

Esempio di richiesta di paginazione

- Tempo di accesso alla memoria = 200 ns
- Tempo medio per il page-fault = 8 ms = 8 000 000 ns
- EAT = (1 – p x 200 + p x 8 000 000)
  = 200 + p x 7 999 800 ns
- **EAT è direttamente proporzionale al tasso p di page fault**
  - **È importante mantenere p basso** per non avere un degrado delle prestazioni del sistema
  - Se ad es. un accesso su 1000 causa un page fault, allora EAT = 200 + 7999,8 ns = 8199,8 ns : un rallentamento di un fattore 40 (200 ns x 40 = 8000 ns)!
  - Per ottenere un rallentamento inferiore al 10% è necessario garantire un page fault ogni 399990 accessi alle pagine

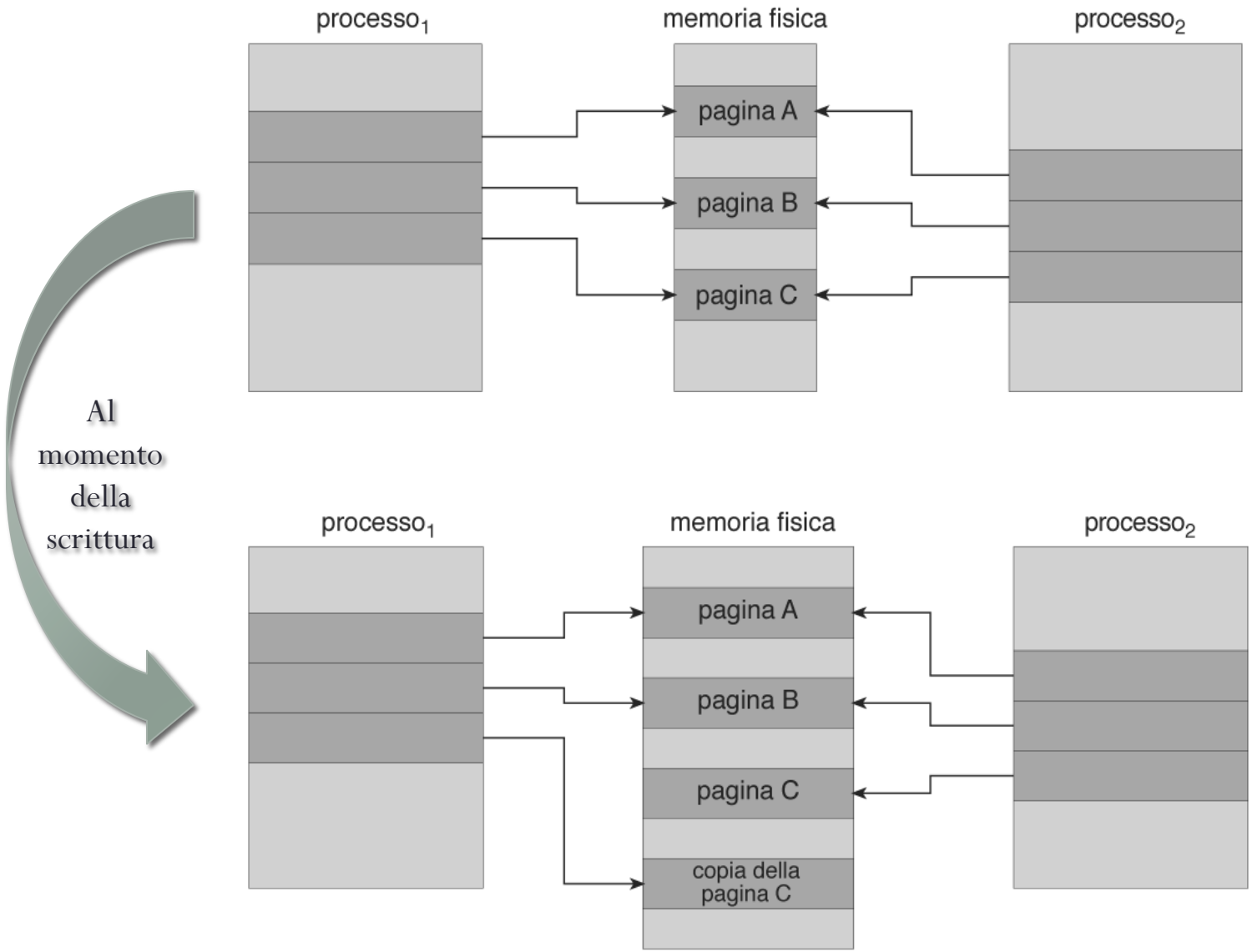
Creazione di un processo

- La memoria virtuale offre altri benefici durante la creazione del processo
- Questi sono resi disponibili attraverso le tecniche:
  - **Copia durante la scrittura** (copy-on-write)
  - **File mappati in memoria**

Copia durante la scrittura (1)

- La chiamata `fork()` crea il processo figlio come un esatto duplicato del padre (copia delle pagine del padre)
- Considerando che molti processi figli eseguono immediatamente la chiamata `exec()` dopo la `fork()`
  - Risulta inutile copiare tutte le pagine
- Per questo motivo viene introdotta la **copiatura su scrittura**
  - Una pagine cosi marcata viene copiata solo quando uno dei due processi la deve modificare
- In questo caso le pagine libere sono tipicamente scelte da un pool di pagine
  - Al momento dell'esecuzione si opera l'**azzeramento su richiesta** (le pagine vengono riempite di zeri)

Copia durante la scrittura (2)



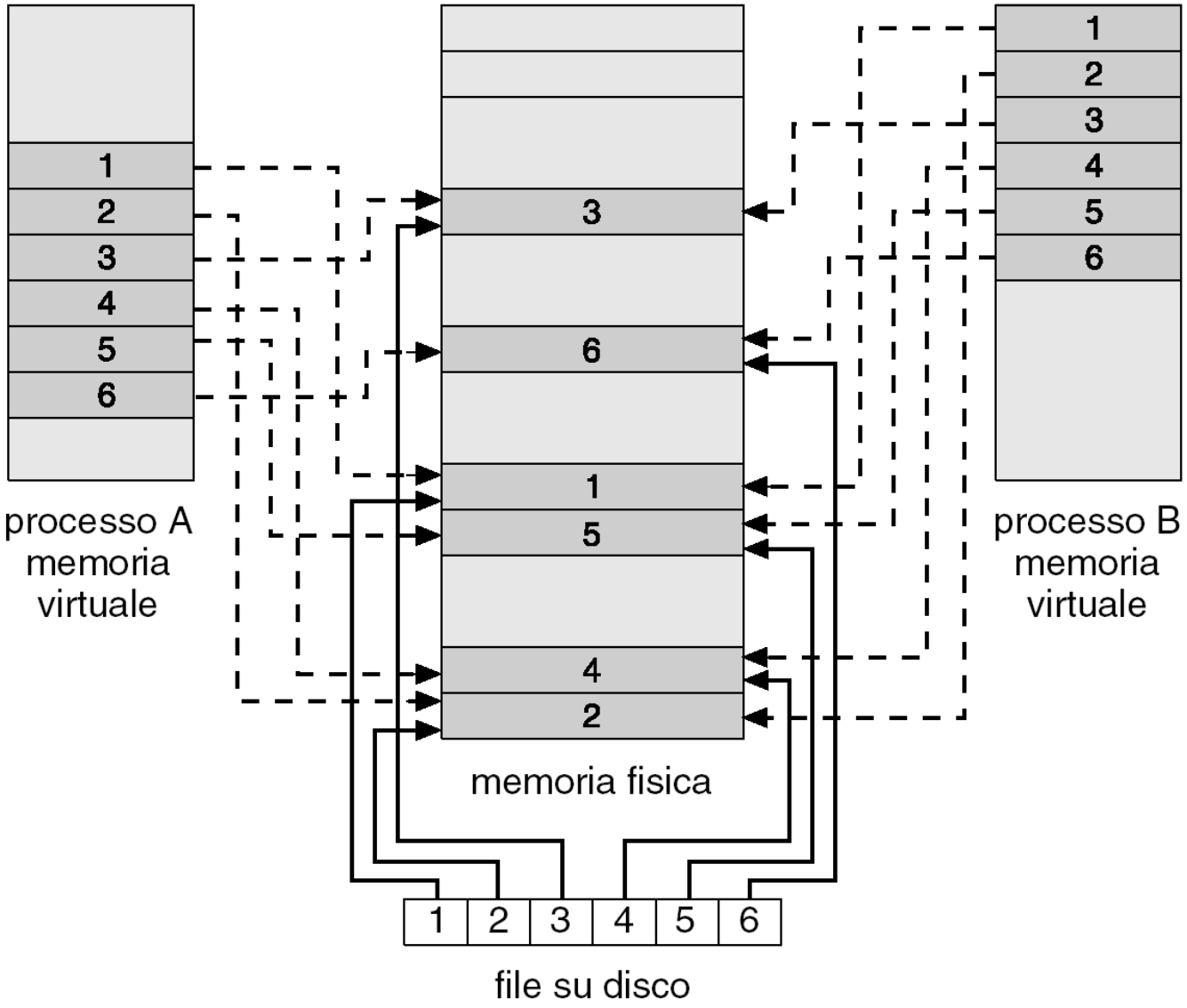
File mappati in memoria (1)

- Si crea una **mappatura tra un file** ed una (o più) pagine in memoria virtuale
- Ogni chiamata di sistema `open()`, `read()`, `write()` può essere sostituita con un accesso alla memoria
  - Molto più semplice e veloce che usare le chiamate di sistema (richiedono il passaggio alla modalità kernel)
- L'accesso iniziale al file (`open()`) consiste in una **richiesta di paginazione**
  - Una porzione del file delle dimensioni di una pagina viene letta dal file system e caricata in una pagina fisica, associata ad una pagina virtuale
- Le successive letture/scritture del file sono trattate come accessi alla memoria centrale

File mappati in memoria (2)

- Le modifiche in memoria possono essere replicate su file periodicamente e alla chiusura del file
- Questo metodo consente anche la condivisione dei file
  - Si mappano più pagine virtuali alla stessa pagina fisica associata al file
- Un file può essere aperto con **copia su scrittura** per avere un **accesso in sola lettura**
- La condivisione dei file richiede l'utilizzo di tecniche di sincronizzazione

File mappati in memoria: condivisione di file



Mappature in memoria dell'I/O

- Convenzionalmente, esistono istruzioni dedicate per il trasferimento dei dati di I/O dai registri del controllore della periferica (controllori video, porte seriali e parallele per modem e stampanti, ecc..) verso la memoria (e viceversa)
- **Mappatura in memoria dell'I/O**: tecnica adottata da molte architetture degli elaboratori per rendere più agevole l'accesso ai dispositivi di I/O

- Alcuni indirizzi di memoria sono riservati per la mappatura dei registri dei dispositivi
  - Periodicamente si trasferiscono i valori dalla memoria ai registri
  - Non è necessario invocare chiamate di sistema

La richiesta di paginazione

- Per implementare la richiesta di paginazione occorre:
  - Un **algoritmo di allocazione dei frame**
    - per decidere quanti frame assegnare ad un processo
  - Un **algoritmo di sostituzione della pagina**
    - per selezionare i frame da sostituire
- Verranno ora analizzati partendo dal secondo

Sostituzione delle pagine

- La memoria fisica è solitamente molto più piccola della memoria logica dei programmi presenti nel sistema
- Col passare del tempo, è molto probabile il verificarsi di un ***page fault*** per il quale **non ci saranno frame liberi** nel sistema
- Soluzione: **sostituire pagine**
  - possibilmente quelle "*che serviranno meno in futuro*"…
  - È conveniente rimpiazzare le pagine in sola lettura o non modificate (con *bit di modifica* a zero)
    - Perché non è necessario copiarle nell'area di swap
  - Diverse *politiche di scelta* di tali pagine

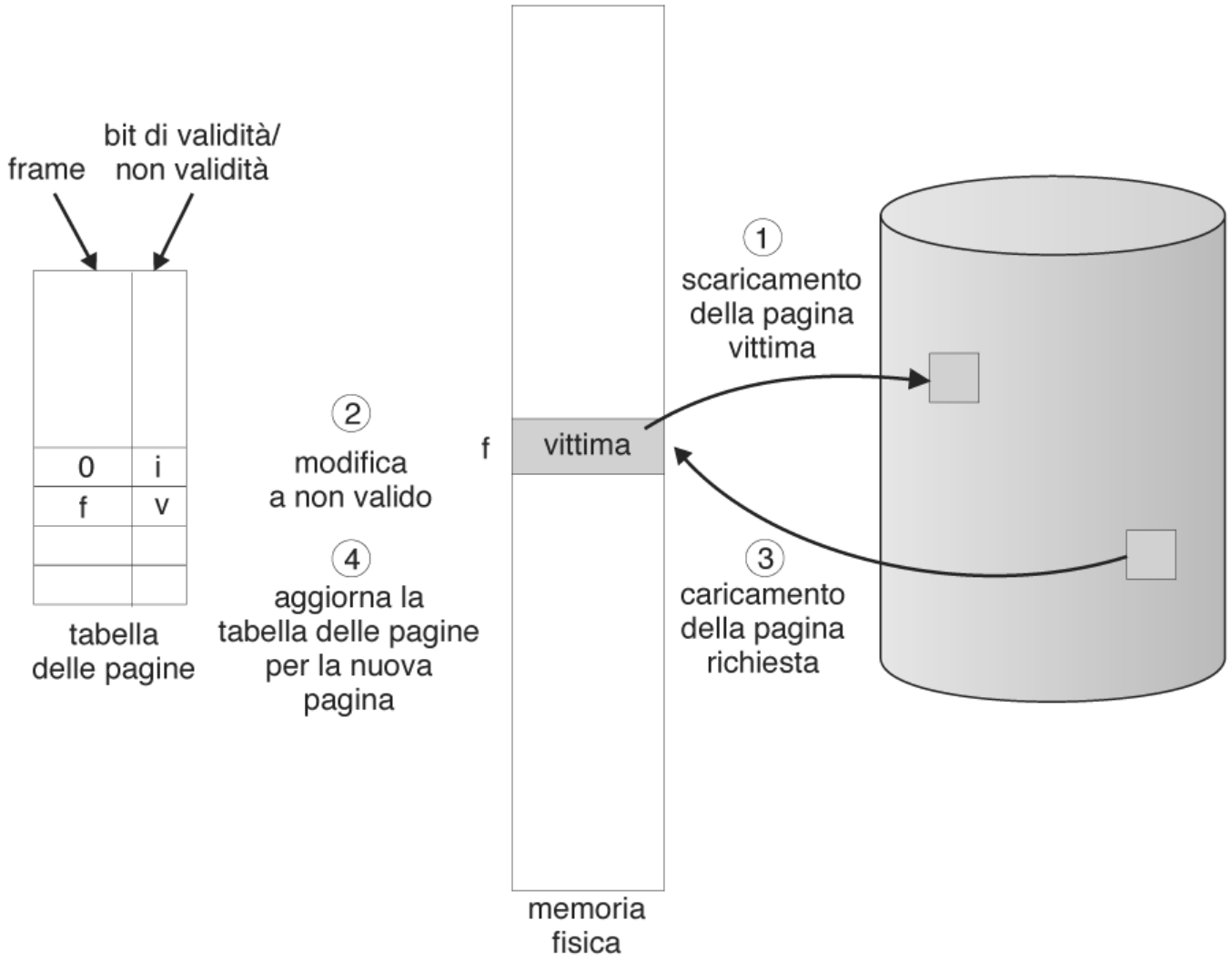
Sostituzione di base della pagina

- Consiste in una modifica della procedura di gestione dell'**eccezione di pagina mancante** (*page fault*)
  1. Si esamina una tabella interna al processo per decidere:
     - Pagina non valida ➔ il processo viene terminato
     - Pagina non in memoria ➔ si procede
  2. Si cerca un frame libero sufficientemente grande
     a. Se esiste lo si usa, altrimenti
     b. Si impiega un'algoritmo per scegliere una pagina vittima da sostituire
     c. Si sposta la pagina vittima sul disco e si aggiornano le tabelle della pagine e dei frame

Modifica

  3. Si caricare dal disco la pagina nel frame liberato
  4. Si modificano la tabella interna al processo e la validità del bit
  5. Si riavvia l'istruzione interrotta dall'eccezione

Passi per la sostituzione della pagina



1. e 3. comportano il trasferimento di due pagine: raddoppia il tempo di servizio della mancanza di pagina e aumenta il tempo EAT (tempo di accesso effettivo)!

Sostituzione di base della pagina

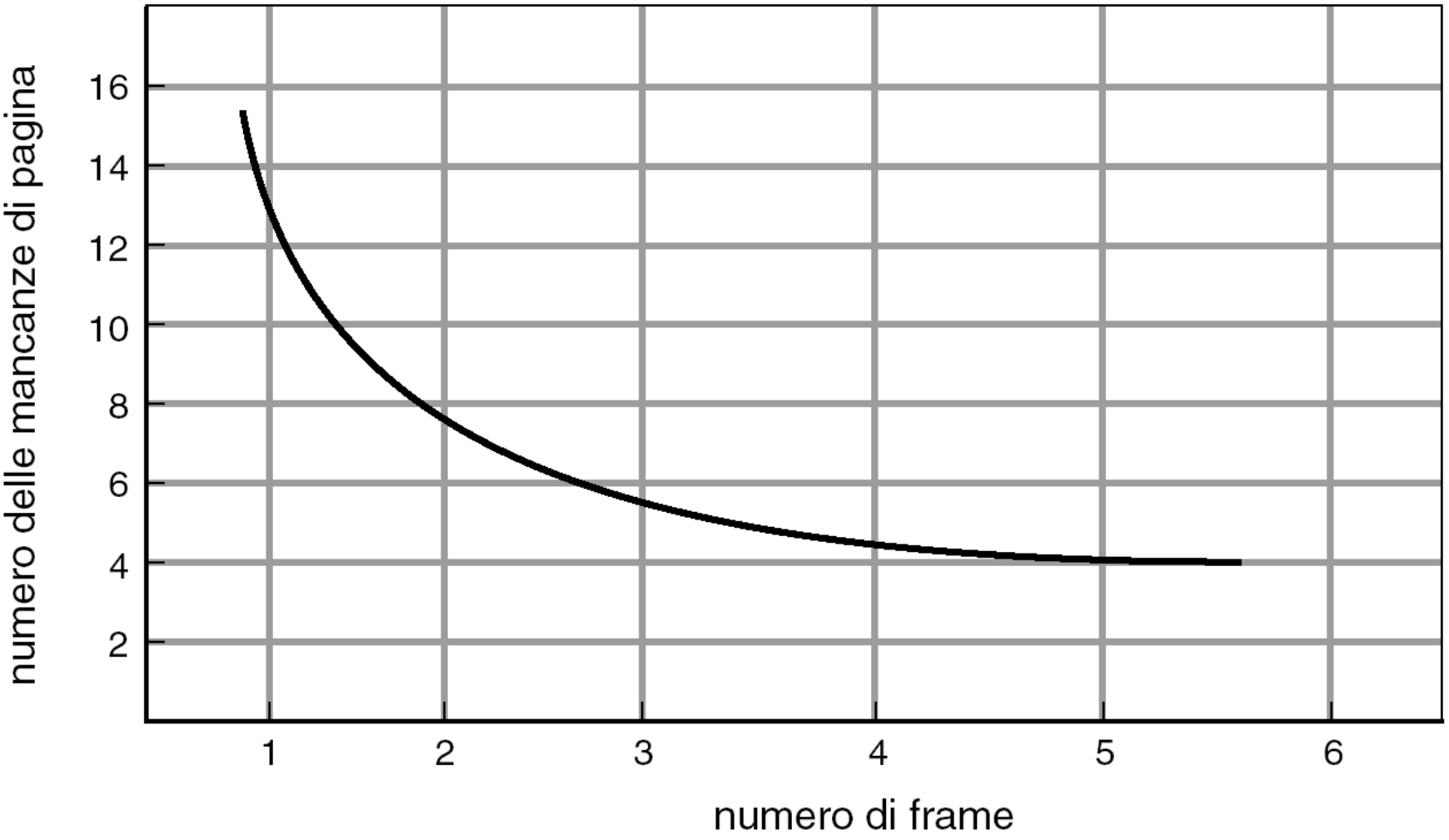
- Le prestazioni della sostituzione di base possono essere migliorate inserendo un **bit di modifica**
  - 1 se la pagina è stata modificata
  - 0 altrimenti (anche per pagine di sola lettura)

- Quando una pagina viene scelta per essere sostituita
  - Se il bit vale 1 deve essere salvata sul disco
  - Altrimenti può essere semplicemente sovrascritta

Valutazione dell'algoritmo di sostituzione della pagina

- **Goal**: desiderare il più basso tasso di mancanza di pagine
- **Un algoritmo viene "valutato"** facendolo operare
  - su esempi di "sequenze di richieste di pagine" (da campioni reali o random): *stringa di riferimento*
  - calcolando il numero di page fault

Grafico delle mancanze di pagina in funzione del numero di frame



Politiche di rimpiazzamento delle pagine (1)

- **First In First Out**
  - Scarica le pagine nel sistema da più tempo

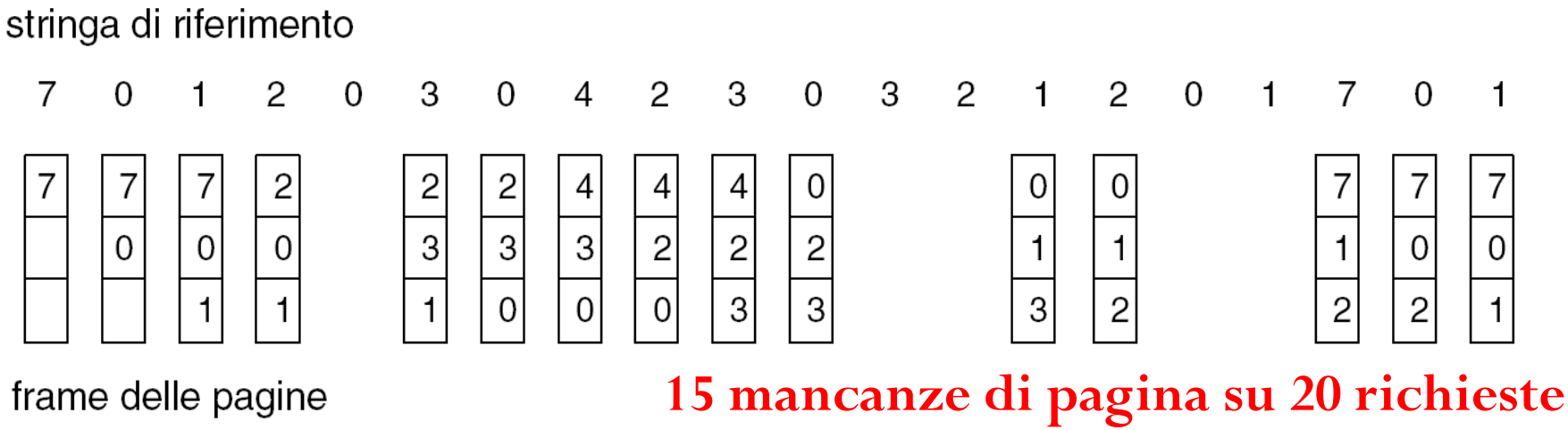
- **FIFO Second Chance**
  - Si applica una politica FIFO fornendo ad ogni pagina una seconda possibilità per restare in memoria

Politiche di rimpiazzamento delle pagine (2)

- **Least Recently Used (LRU)**
  - Scarica le pagine non usate da più tempo
  - Ad ogni pagina è associato un timer di "vecchiaia"
- **Not Recently Used (NRU)**
  - Scarica le pagine non usate di recente
  - Si basa sui bit usata e modificata

Sostituzione FIFO della pagina

- È l'algoritmo più semplice
- Si sceglie la pagina che è stata caricata da più tempo
  - Timer associato ad ogni pagina per il tempo di caricamento, o
  - Una coda: sostituzione in cima, caricamento in coda
- Esempio: memoria fisica di 3 frame inizialmente vuoti
  - I frame sono riportati ogni volta che c'è una sostituzione di pagina



15 mancanze di pagina su 20 richieste

Sostituzione FIFO della pagina (cont.)

- Stringa di riferimento: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- **3 frame**
  - 9 mancanze

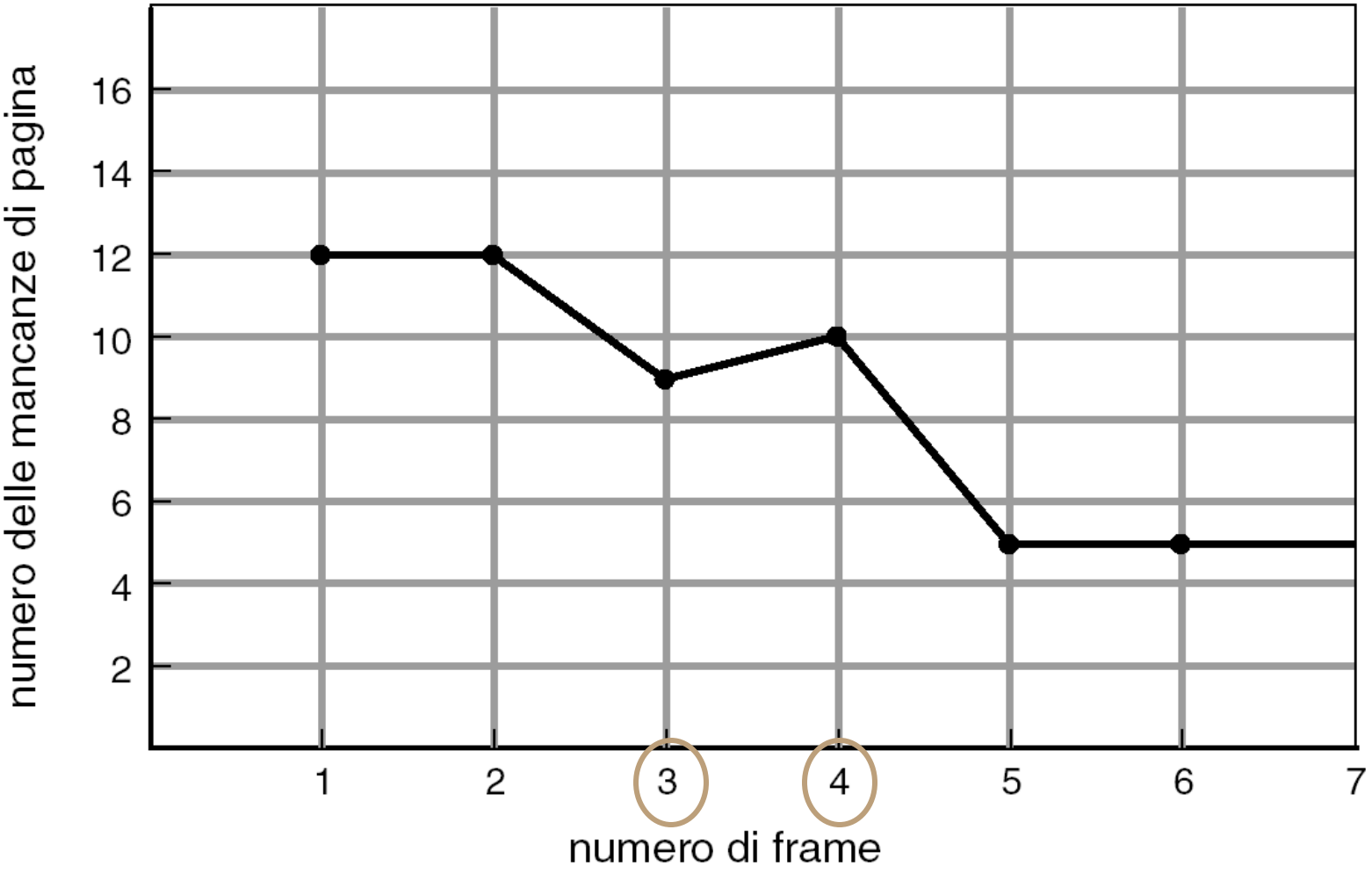
1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5			5	5	
	2	2	2	1	1	1			3	3	
		3	3	3	2	2			2	4	

- **4 frame**
  - 10 mancanze

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1			5	5	5	5	4	4
	2	2	2			2	1	1	1	1	5
		3	3			3	3	2	2	2	2
			4			4	4	4	3	3	3

- Sostituzione FIFO – **Anomalia di Belady**
- Aumentando i frame aumentano le mancanze di pagina!

Sostituzione FIFO che illustra l'anomalia di Belady



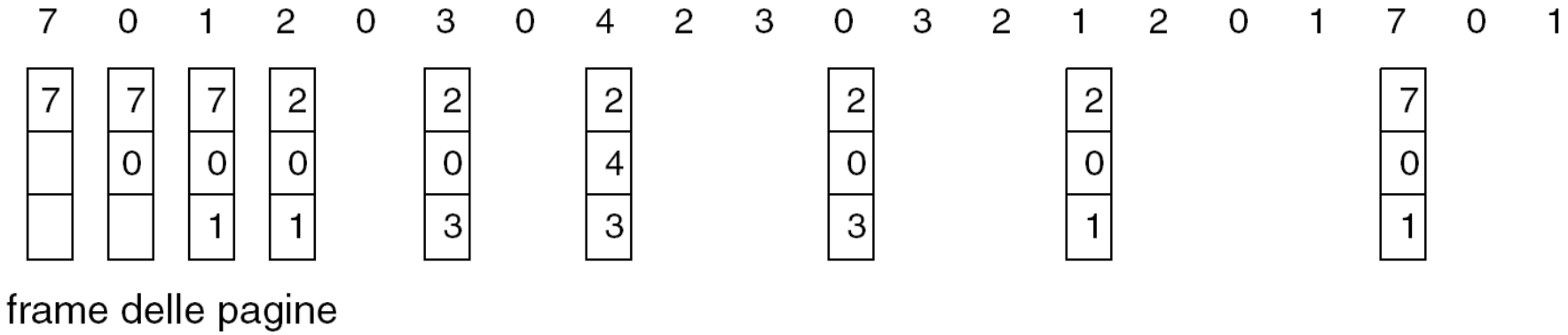
Algoritmo ottimale (1)

- È l'algoritmo tale per cui
  - Il numero di mancanze di pagina è il minore possibile
  - Non presenta l'anomalia di Belady

- **Pagina vittima**: Sostituire la pagina che **non sarà usata per il più lungo periodo di tempo**

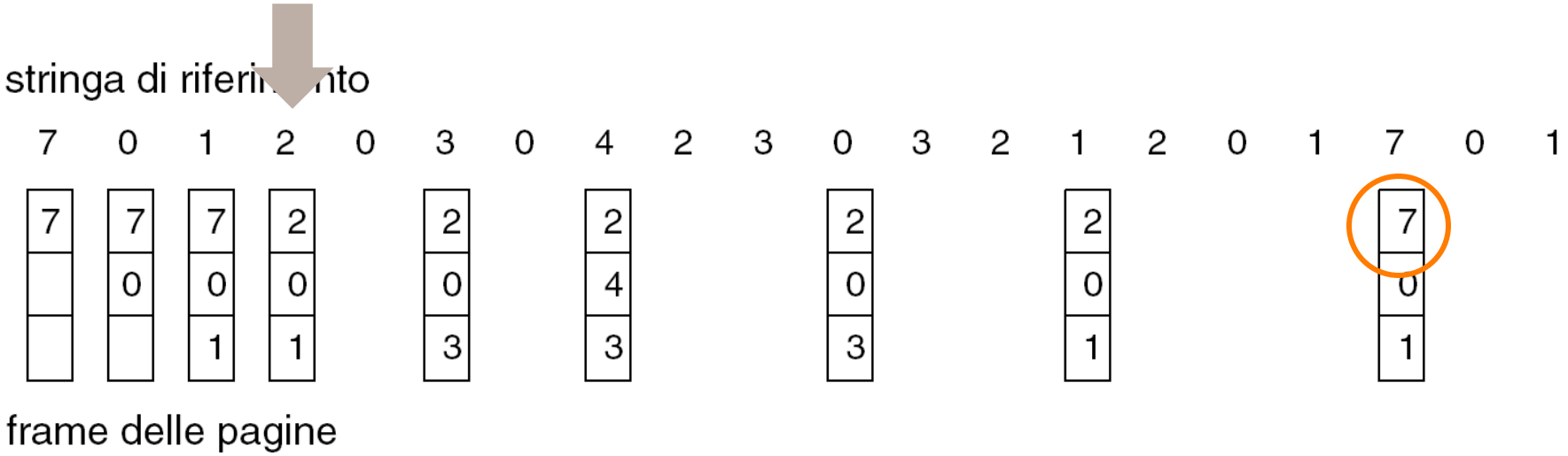
Algoritmo ottimale (2)

stringa di riferimento

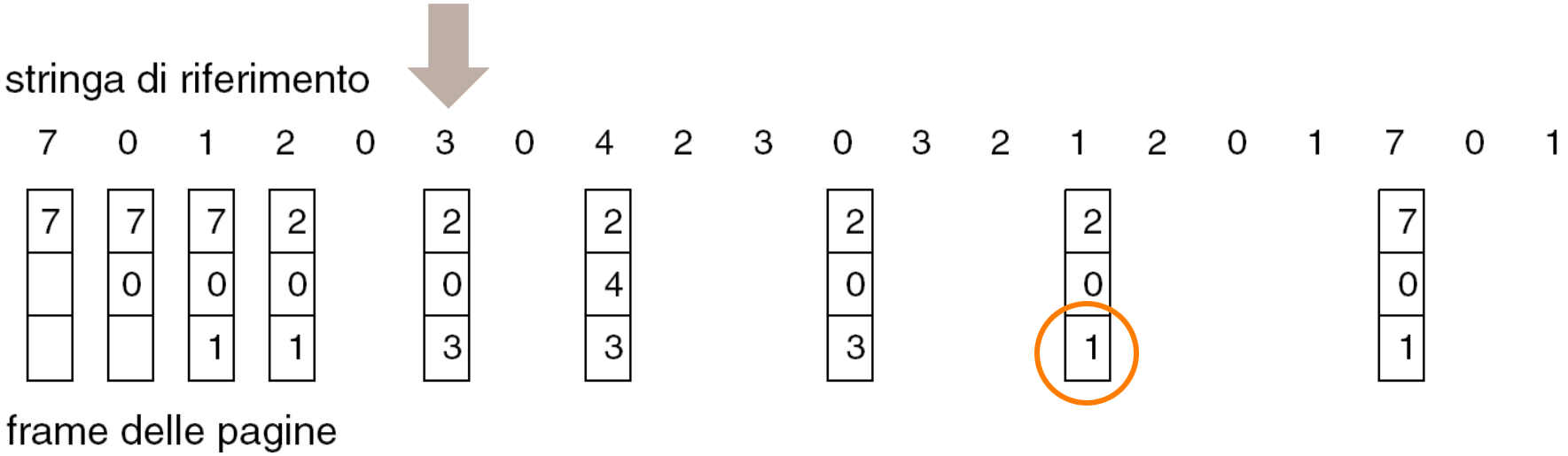


frame delle pagine

Algoritmo ottimale (2)



Algoritmo ottimale (2)



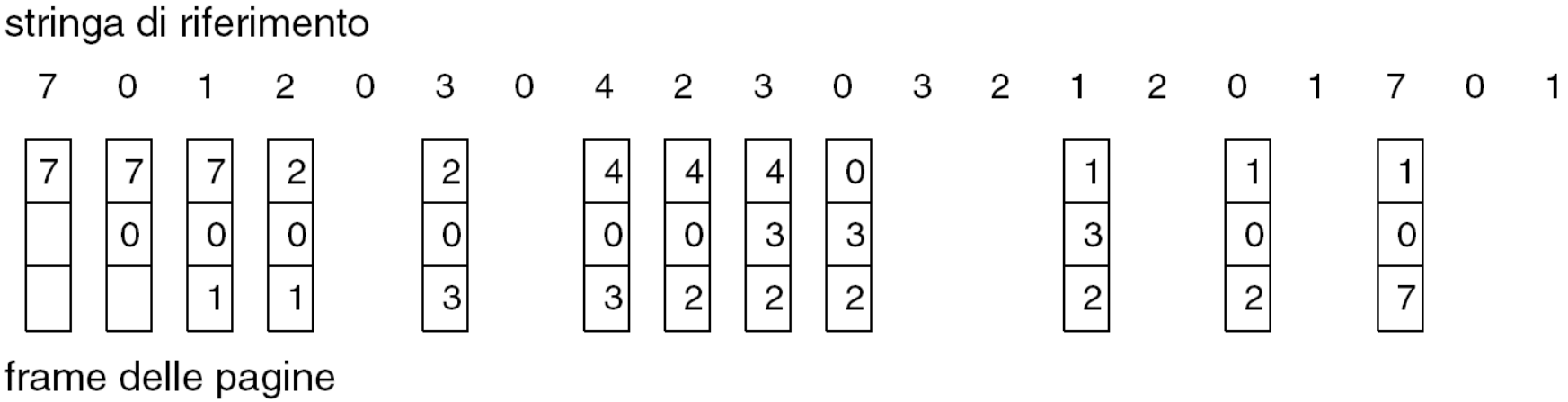
- **9 mancanze di pagina contro le 15 di FIFO**
- **Se si escludono le prime 3 che non dipendono dall'algoritmo: 6 vs 12 (50 % più veloce)**

Algoritmo ottimale (3)

- **Problema**: è difficile stabilire quali pagine non saranno usate per più a lungo delle altre
- Per questo motivo questo algoritmo è ottimale ma anche **IDEALE** (difficilmente implementabile)
- Tipicamente si usa come **algoritmo di riferimento** per valutare algoritmi non ideali

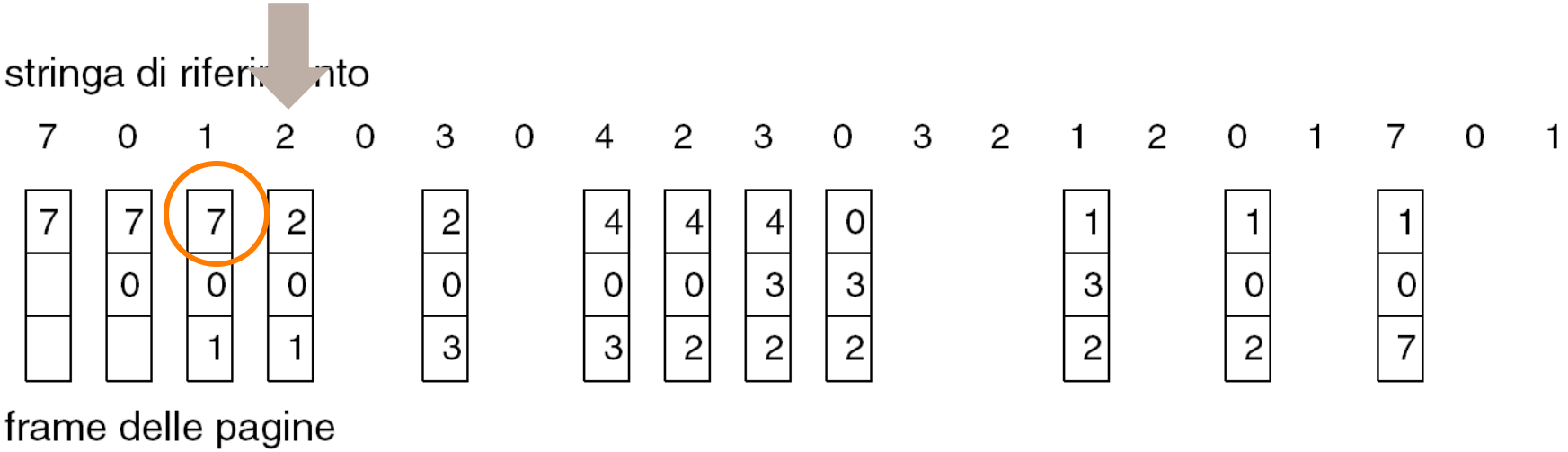
Algoritmo LRU (Least-Recently-used)

- **Pagina vittima**: sostituire la pagina che **non è stata usata per il periodo di tempo più lungo**



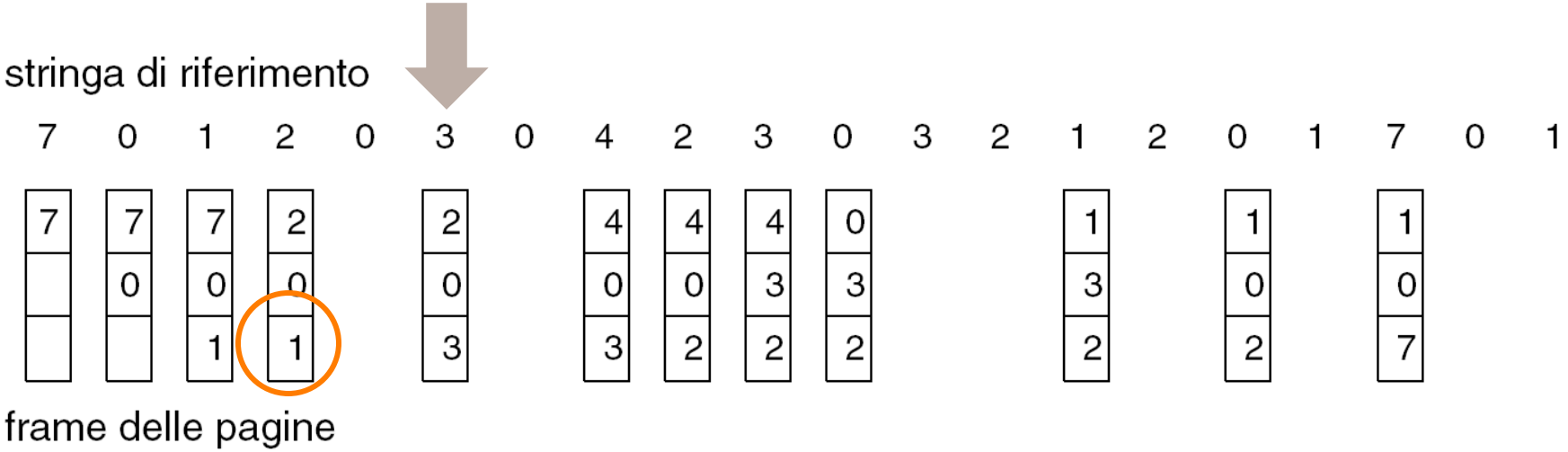
Algoritmo LRU (Least-Recently-used)

- **Pagina vittima**: Sostituire la pagina che **non è stata usata per il periodo di tempo più lungo**



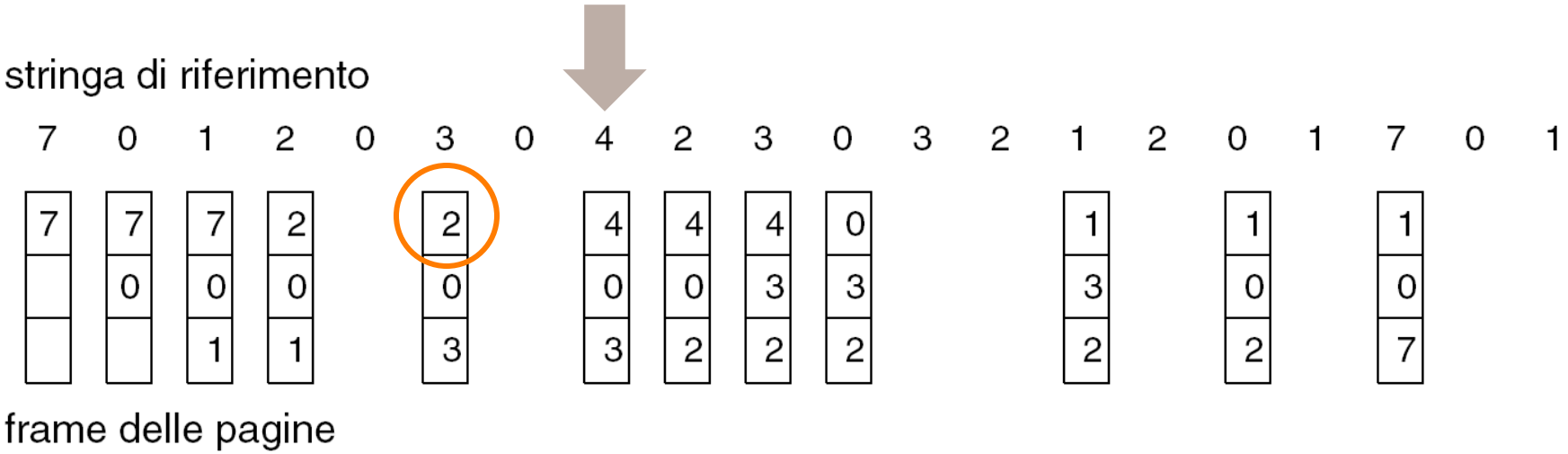
Algoritmo LRU (Least-Recently-used)

- **Pagina vittima**: Sostituire la pagina che **non è stata usata per il periodo di tempo più lungo**



Algoritmo LRU (Least-Recently-used)

- **Pagina vittima**: Sostituire la pagina che **non è stata usata per il periodo di tempo più lungo**



12 mancanze di pagina su 20 richieste

Implementazione dell'algoritmo LRU

- Non mostrano mai l'anomalia di Belady
- Il problema è determinare un ordine dei frame, dal momento del "loro ultimo uso"

- Può richiedere supporto hardware

- Due possibili implementazioni:
  - tramite **contatore di vecchiaia**
  - tramite **stack** (o pila)

Implementazione dell'algoritmo LRU
Contatore di vecchiaia

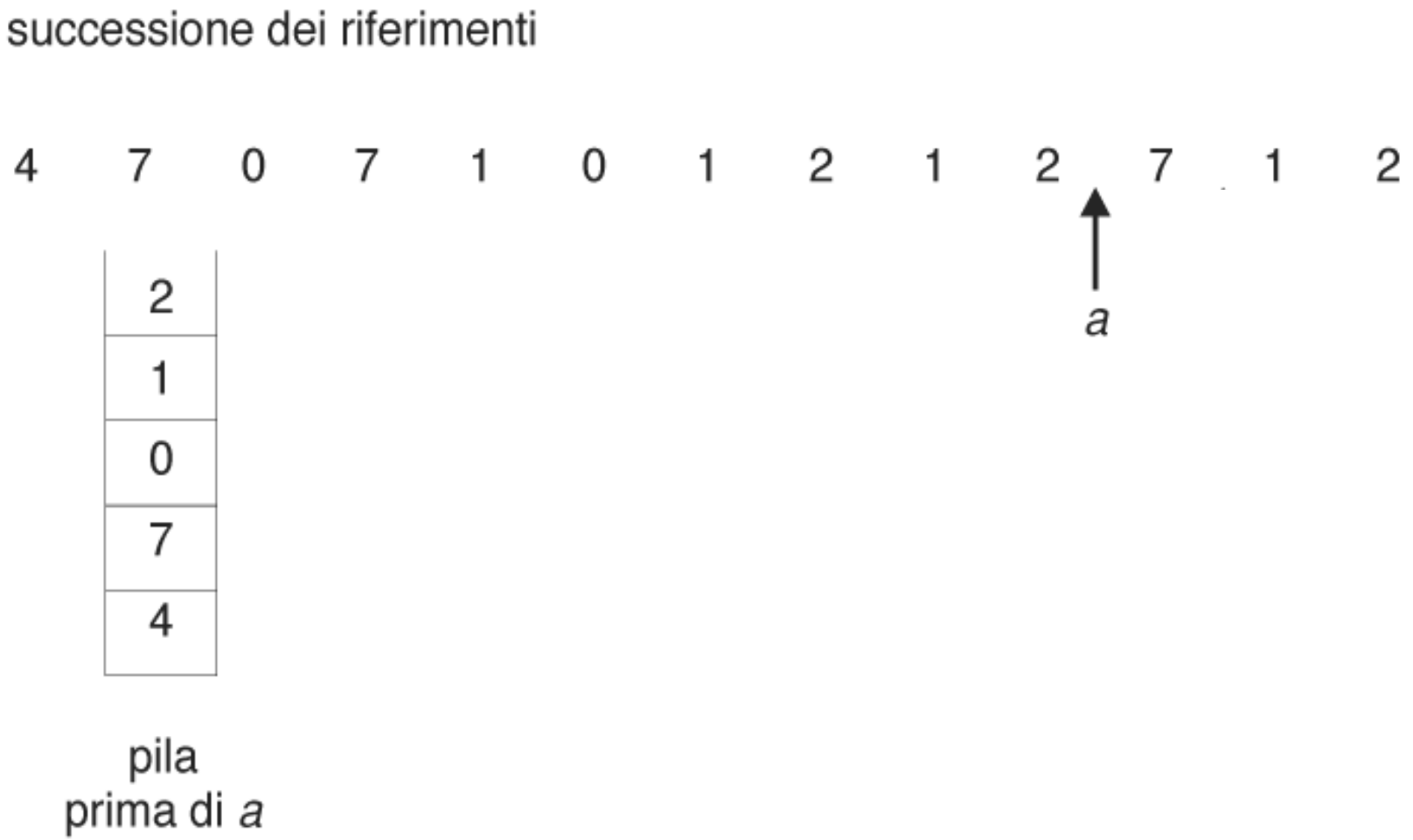
- Si aggiunge alla CPU **un orologio** o un contatore logico
  - Incrementato ad ogni accesso alla memoria
- Nella tabella PT ogni pagina ha un registro nel quale può essere salvato il valore del contatore
- Ad ogni accesso ad una pagina il valore del suo registro **viene aggiornato scrivendo il valore del contatore**
- Si sostituisce la pagina con valore del registro minore

- Richiede una ricerca nella PT ed una scrittura in memoria per ogni accesso alla memoria
- Bisogna considerare il problema dell'overflow dell'orologio

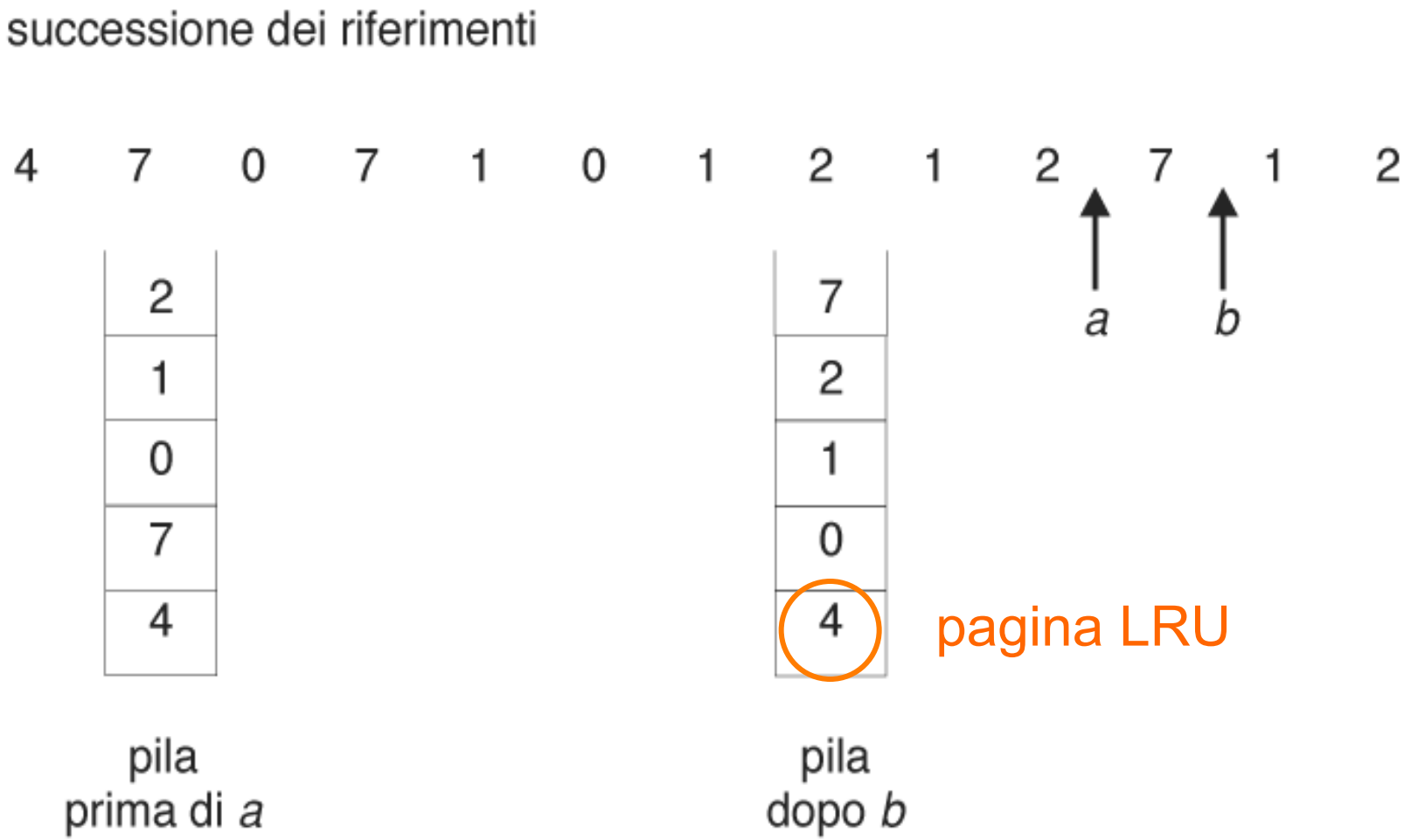
Implementazione dell'algoritmo LRU Stack

- Si conserva uno stack contenente i numeri della pagine
- Ogni volta che una pagina viene usata si estrae il suo numero dallo stack o lo si pone in cima
- In fondo alla pila quindi si trova sempre l'ultima pagina usata
- Implementazione: una lista a doppio collegamento con puntatore all'elemento iniziale e finale
- L'aggiornamento della pila è più costoso che quello del contatore
- Ma la sostituzione non richiede una ricerca
  - Si usa l'elemento in fondo allo stack

Uso di uno stack per registrare i riferimenti alle pagine usate più di recente



Uso di uno stack per registrare i riferimenti alle pagine usate più di recente



Approssimazione dell'algoritmo LRU (1)

- In verità, entrambe le due implementazioni (contatori, stack) appesantiscono il sistema
- Perché l'aggiornamento dei campi orologio o dello stack deve essere eseguito per ogni riferimento alla memoria
- Ciò richiede un segnale di interruzione che rallenta di circa 10 volte gli accessi alla memoria
- Questo problema può essere ininfluente nella TLB (in quanto molto veloce) ma non nelle tabelle contenute in memoria

Approssimazione dell'algoritmo LRU (2)

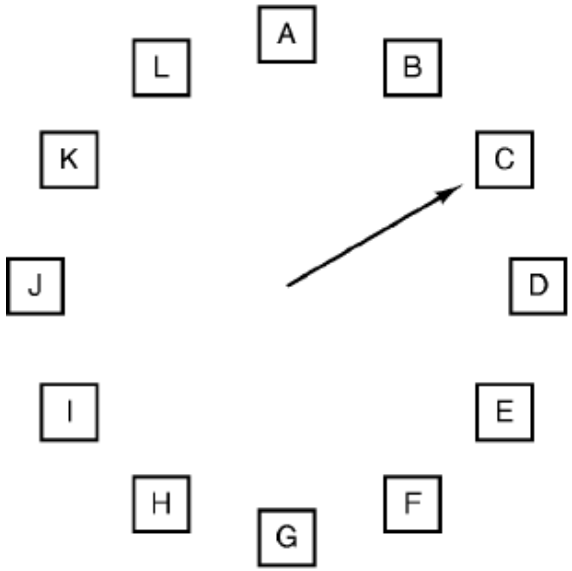
- Molti computer, forniscono una soluzione HW
- Il **bit di riferimento** (o usata)
  - Ad ogni pagina è associato un bit, inizialmente posto a 0
  - Quando la pagina è referenziata (in lettura o scrittura) il bit è impostato a 1
  - Esistono diversi algoritmi che usano questo bit per approssimare la politica LRU
    - Differiscono nel modo in cui sono implementati

Approssimazione dell'algoritmo LRU (3)

- **Algoritmo con bit supplementari di riferimento**
  - Per ogni pagina si conserva in memoria un vettore di dimensione n
  - Periodicamente il bit di riferimento viene spostato nella cella più a sinistra del vettore (bit più significativo)
    - I bit delle celle del vettore vengono shiftati a dx e il bit dell'ultima viene scartato (bit meno significativo)
    - Il bit viene poi azzerato
  - In ogni istante la pagina il cui vettore contiene il numero binario più piccolo è quella usata meno di recente
    - Possono esserci più pagine con lo stesso valore
    - Si sostituiscono tutte o si applica FIFO

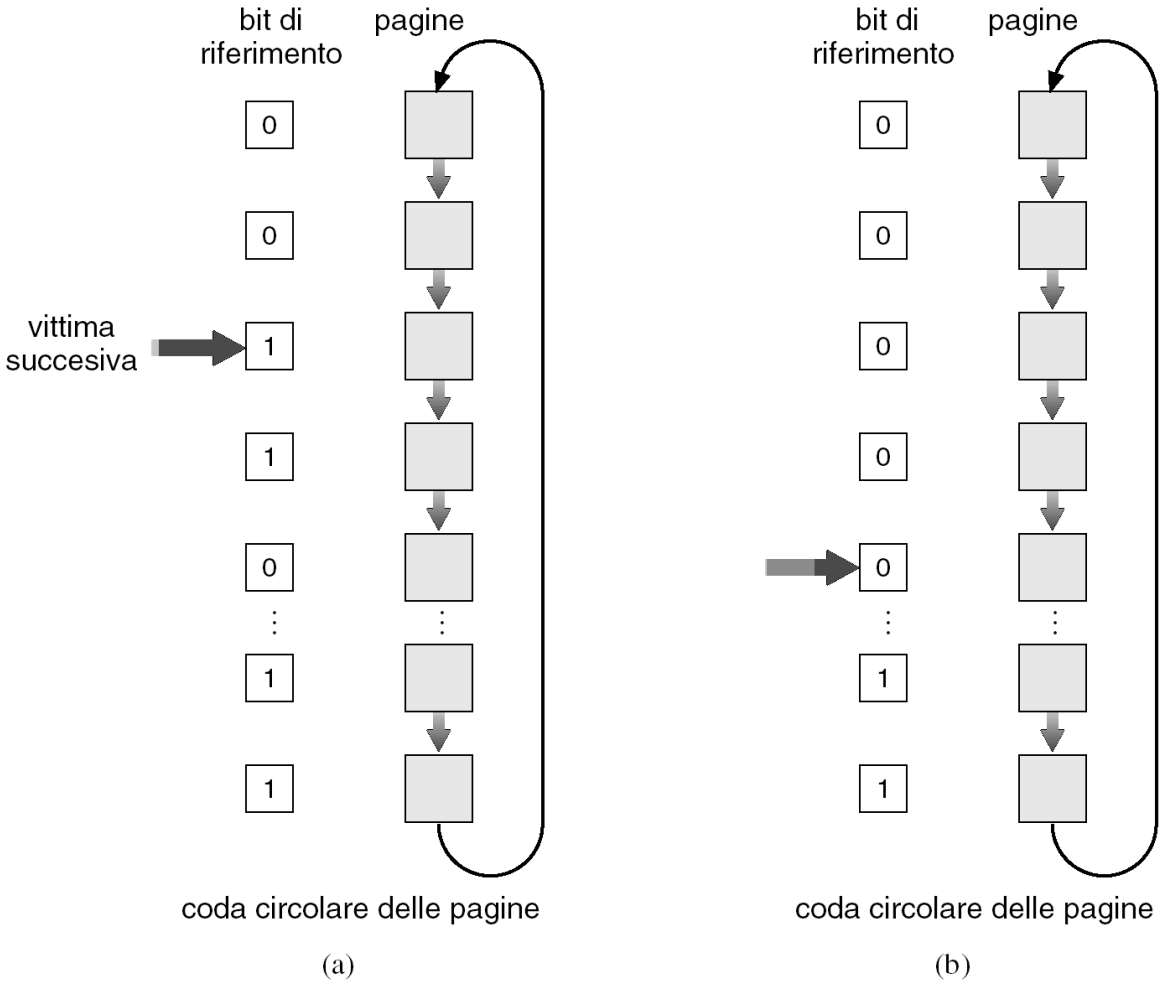
Approssimazione dell'algoritmo LRU (4)

- **Algoritmo seconda possibilità (o dell'orologio)**
  - Le pagine sono disposte in **una lista circolare**
  - Quando occorre selezionare una pagina vittima, la lista viene scansionata partendo dall'ultima pagina analizzata:
    - se una pagina ha il bit di riferimento a 1
      - viene posto a 0 e si passa alla pagina successiva
    - se una pagina ha il bit di riferimento a 0
      - Si sceglie la pagina per la sostituzione



Approssimazione dell'algoritmo LRU (5)

Algoritmo della seconda possibilità (orologio)



Approssimazione dell'algoritmo LRU (6)

- **Algoritmo seconda possibilità migliorato o algoritmo NRU (Not Recently Used)**

  - Utilizza sia il bit di riferimento che il bit modificata
    - come coppia ordinata (R,M)
  - Le pagina sono cosi raggruppate in 4 classi:
    1. (0,0) non usate e non modificate
    2. (0,1) non usate e modificate (dovranno essere scritte in memoria prima di essere sostituite)
    3. (1,0) usate e non modificate (saranno probabilmente usate di nuovo a breve)
    4. (1,1) usate e modificate

Approssimazione dell'algoritmo LRU (7)

- **Algoritmo seconda possibilità migliorato o algoritmo NRU (Not Recently Used)**
  - L'algoritmo scarta la prima pagina fra quelle appartenenti alla classe più bassa e non vuota
    - È necessario fare più di un giro
      - Il primo per classificare le pagine
      - Il secondo per raggiungere la vittima

Altri algoritmi: algoritmi di conteggio

- Si basano sull'idea di mantenere un contatore del numero di riferimenti che sono stati fatti ad ogni pagina
- **Algoritmo LFU (Least Frequently Used)**: sostituisce la pagina con il più basso conteggio
  - Idea: pagine usate attivamente devono avere un conteggio alto
  - Problema: pagine molto usate in fase di inizializzazione e non più usate
- **Algoritmo MFU (Most Frequently Used)**: sostituisce la pagina con il più alto conteggio
  - Idea: pagina con il conteggio più basso è stata probabilmente appena caricate e deve ancora essere usata

- Poco usati (implementazione costosa) e non approssimano bene l'algoritmo ottimale

Allocazione dei frame

- Problema: quante e quali pagina devono essere caricate nel momento in cui il processo viene lanciato?
- Per aumentare l'efficienza è bene caricare un **numero minimo di frame**
  - Legato dall'architettura del computer (*instruction-set*)
  - È il numero minimo che permette l'esecuzione di un'istruzione senza generare un'eccezione di pagina mancante

- **Esempio: IBM 370**: 6 pagine per gestire l'istruzione MVC per muovere caratteri da memoria a memoria:
  - Istruzione a 6 byte, può occupare 2 pagine
  - 2 pagine per gestire il blocco di caratteri "da" muovere
  - 2 pagine per gestire la zona "verso" cui muovere
  - È quindi necessario allocare almeno 6 frame per poter eseguire l'istruzione

Algoritmi di allocazione dei frame

- Definito il numero minimo bisogna scegliere quanti frame associare ad ogni processo

- Due principali schemi di allocazione:
  - **Allocazione uniforme**
  - **Allocazione non uniforme**

Allocazione uniforme

- **Allocazione omogenea**: si assegna lo stesso numero di frame ad ogni processo: per esempio, se si hanno 100 frame e 5 processi, ognuno prende 20 pagine
- **Allocazione proporzionale**: si assegna la memoria disponibile ad ogni processo in base alle dimensioni di quest'ultimo

$$s_i = \text{dimensione del processo } p_i$$

$$S = \sum s_i$$

$$m = \#\text{ totale dei frame}$$

$$a_i = \text{spazio allocato a } p_i = \frac{s_i}{S} \times m$$

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Allocazione a priorità

- Si assegna ai processi a priorità più alta più memoria (indipendentemente dalla loro dimensione)
- Si usa uno schema di allocazione **proporzionale** che
  - **usa le priorità** piuttosto che la dimensione
  - o una combinazione delle due

Allocazione globale e locale

- Un'altra questione riguarda la sostituzione delle pagine:
- **Sostituzione locale**: ogni processo effettua la scelta solo nel proprio insieme di frame allocati
- **Sostituzione globale**: permette ad un processo di selezionare un frame di sostituzione a partire dall'insieme di tutti i frame
  - Anche se quel frame è correntemente allocato a qualche altro processo
  - Vantaggioso per favorire processi a priorità alta

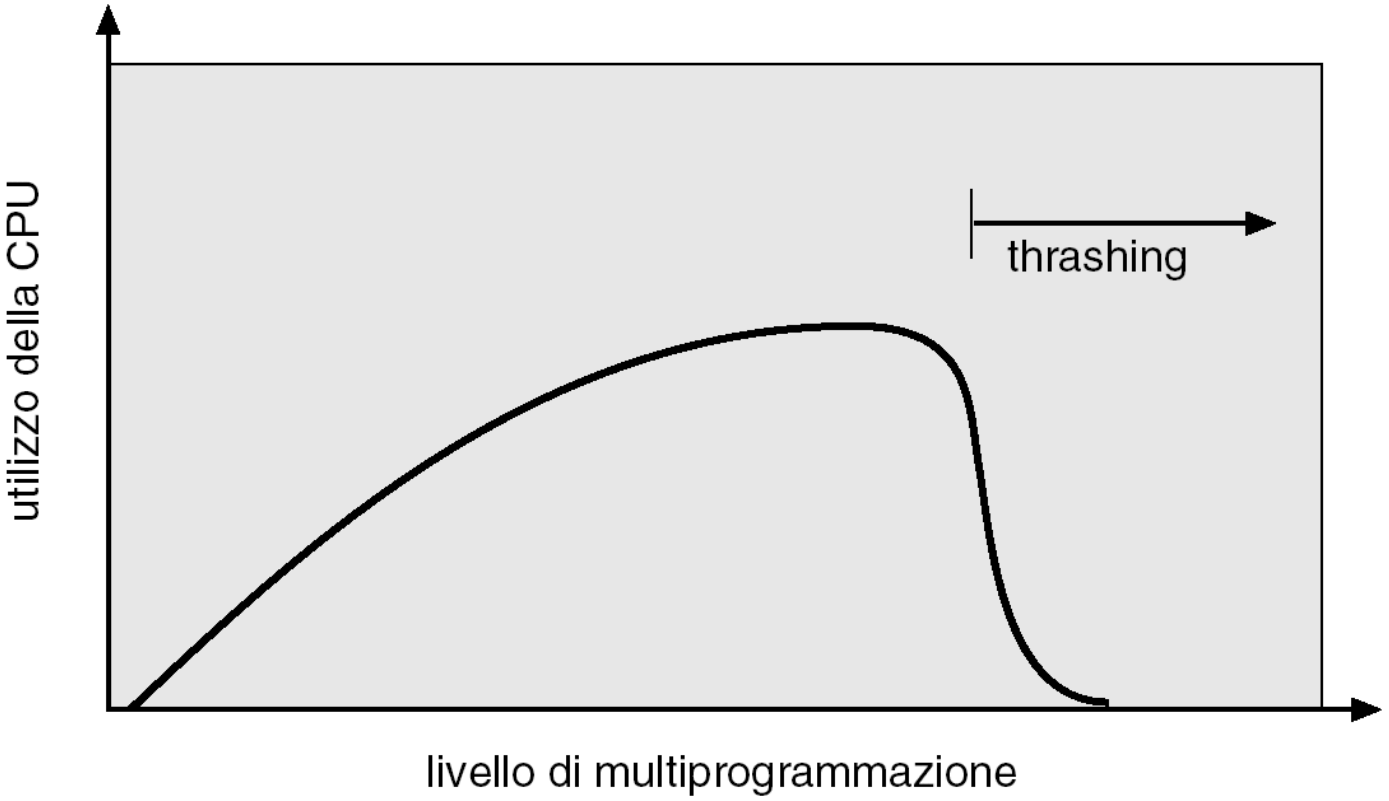
- **La sostituzione globale è la più usata**, perché da un miglior rendimento del sistema
- D'altra parte però il tasso di page-fault non dipende solo dal comportamento del processo ma anche dagli altri

Thrashing (paginazione degenere)

- **Thrashing**: fenomeno tale per cui un processo spende **più tempo nella paginazione che** nella propria esecuzione
- Le cause possono essere spiegate illustrando il funzionamento dei primi sistemi di paginazione (non usavano tecniche per prevenire il trashing):
  1. Se un processo non ha un numero di pagine inferiore a quelle necessarie, il tasso di mancanza di pagina cresce
  2. Il processo (in uno schema di allocazione globale) prenderà i frame degli altri processi e similmente faranno altri processi
     - Viene messo nella coda del dispositivo di paginazione
  3. Se molti processi entrano in questa coda la coda dei processi pronti si svuota velocemente
     - L'efficienza della CPU cala, quindi il SO aumenta il livello di multiprogrammazione
  4. I nuovi processi hanno bisogno di acquisire frame, che però non sono liberi
     - Entrano quindi nella coda del dispositivo di paginazione
  5. Si torna al punto 3

Thrashing e il livello di multiprogrammazione

- Il livello di multiprogrammazione aumenta l'efficienza fino ad un certo punto
  - Dopodiché entra in gioco la paginazione degenere



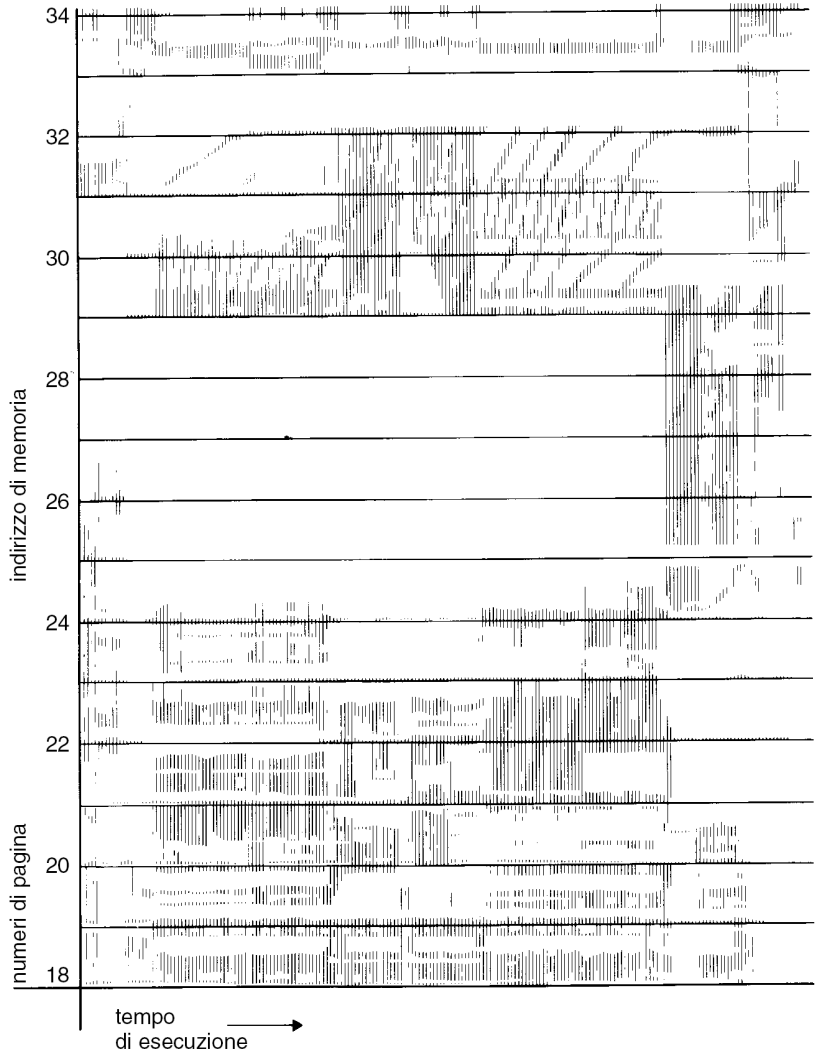
Limitare il Thrashing

- Per evitare il trashing bisogna assicurare che un processo abbia sempre il numero di frame necessari
  - Quanti?

- Esistono diverse tecniche
- La **strategia del "working set"** inizia osservando quanti frame sta attualmente usando un processo
- Al fine di definire un **modello di località**

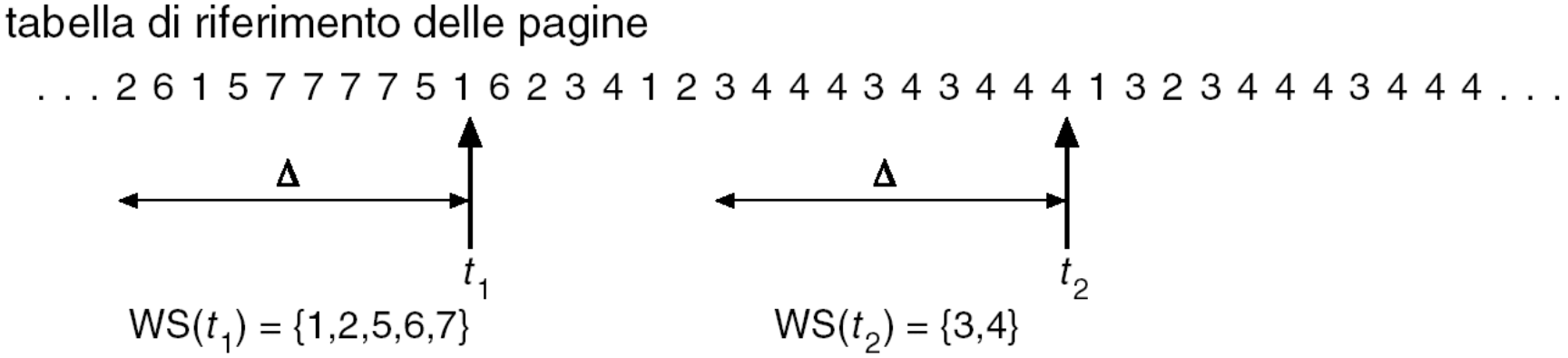
Località in un modello di riferimento alla memoria

- Un programma è generalmente composto da varie località (insieme di *pagine attive*)
- Quando un processo chiama una procedura crea una nuova località
  - Contiene istruzioni e variabili locali
- La struttura del programma definisce quindi le località
- Il trashing si verifica se:
  - **La somma delle dimensione della località > dimensione totale della memoria fisica allocata**



Il modello del working-set (1)

- Il modello del working-set usa un parametro Δ per definire la dimensione della finestra dell'insieme di lavoro (working-set)
- Idea: si esamino i più recenti **Δ riferimenti alle pagine**
  - Le pagine che sono state accedute in lettura o scrittura in questa finestra di dimensione Δ fanno parte del working-set
- Il working-set è quindi un'approssimazione delle località del programma



Il modello del working-set (2)

- La dimensione di Δ influenza la precisione dell'algoritmo:
  - Troppo basso: il working-set non include l'intera località
  - Troppo alto: più località si sovrappongono
  - Se tende ad infinito il working-set coincide con tutte la pagine usate dal processo nella sua esistenza
- Una volta noto il working-set è possibile calcolare la richiesta totale di frame D

$$D = \sum_{i=1}^{n} WSS_i$$

  - Dove: WSS_i è il numero di pagine referenziate dal processo P_i durante gli ultimi Δ riferimenti
- **D deve tale per cui: D < m** (# totale di frame liberi)
  - Altrimenti si verifica il fenomeno della paginazione degenere

Il modello del working-set (3)

- Verificato che $D < m$
  - Il SO assegna ad ogni processo il numero di frame WSS_i
  - Se rimangono sufficienti frame liberi è possibile avviare un nuovo processo
- Se invece $D > m$
  - Il SO individua uno o più processi da sospendere
  - Salva le pagine del processo in memoria di massa
  - Il processo riprenderà successivamente

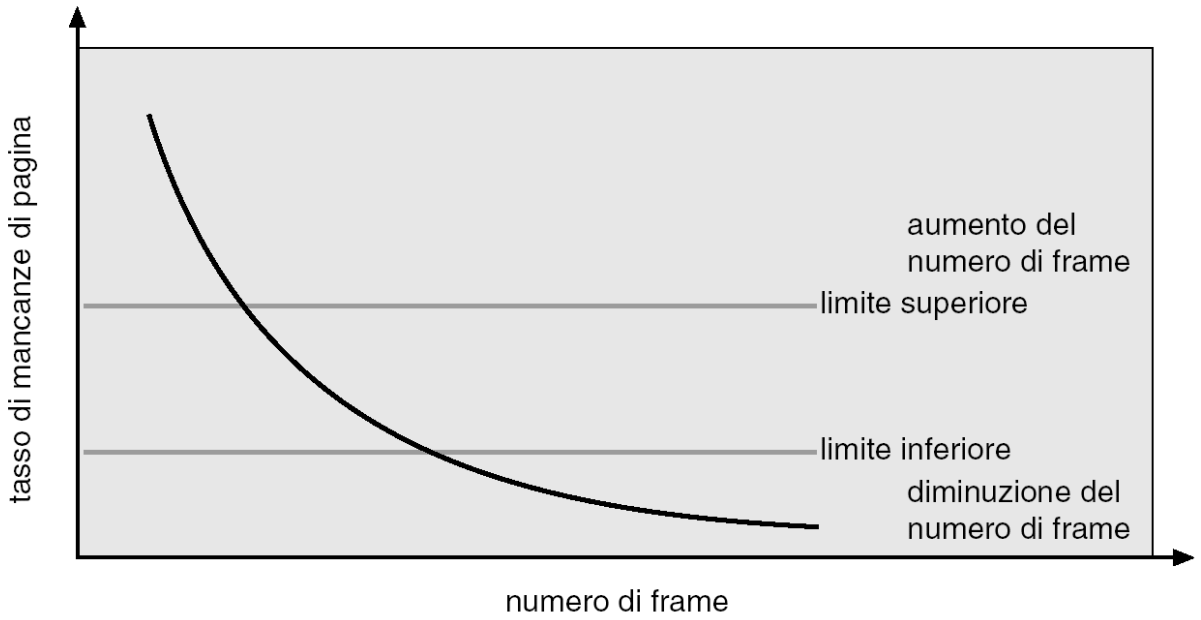
- In questo modo il SO garantisce che la paginazione non degeneri e inoltre mantiene il livello di multiprogrammazione più alto possibile
  - Ottimizza l'uso della CPU

Mantenere traccia del working set

- Si può approssimare il modello working set con un interrupt di **un temporizzatore a intervalli fissi di tempo** e **un bit di riferimento (usata)**
- Esempio: Δ = 10.000 riferimenti e interrupt di temporizzatore ogni 5000 riferimenti
  - Tenere in memoria 2 bit per ogni pagina
    - Ogni volta che si riceve l'interrupt del temporizzatore, si salvano in memoria e si azzerano i valori del bit di riferimento per ogni pagina
  - Usando il bit di riferimento e i 2 bit di memoria si può stabilire se la pagina era in uso nell'intervallo 0-15.000
  - Se uno dei bit vale = 1 ➔ la pagina è nel working set
  - Non del tutto preciso. Se vogliamo aumentare la precisione:
    - Cronologia = 10 bit e interrupt ogni 1000 riferimenti

Frequenza delle mancanze di pagina

- Metodo più diretto per controllare il trashing
- **Stabilire un tasso accettabile per la frequenza di mancanze di pagina**:
  - Se il tasso attuale è troppo basso, si deallocano alcuni frame del processo
  - Se il tasso attuale è troppo alto, si allocano nuovi frame al processo



Altre considerazioni

- Prepaginazione
- Dimensione delle pagine
- Portata della TLB
- Struttura dei programmi
- Vincolo di I/O
- Elaborazione in tempo reale

Prepaginazione (1)

- Tecnica **per prevenire un elevato numero di assenze di pagina** che si verificano
  - Quando un processo viene ricaricato in memoria dopo essere stato sospeso (swap in)
- Ad ogni processo viene associata una lista delle pagine contenute nel suo working set
- Quando il processo deve essere sospeso (e.g. I/O o assenza di frame liberi) questa lista viene salvata
- Prima di riavviare il processo (completamento I/O, frame liberi)
  - Tutte le pagine del precedente working-set vengono ricaricate in memoria

Prepaginazione (2)

- Le prepaginazione è conveniente nel caso la maggior parte delle pagine ricaricate in memoria saranno realmente riutilizzate

- Se vengono prepaginate s pagine
  - αs: risparmio dovuto alle eccezioni di pagina mancante evitate grazie alla prepaginazione
  - $(1-\alpha)s$: costo di prepaginazione delle pagine non riutilizzate

- $\alpha \rightarrow 0$: prepaginazione non conveniente
- $\alpha \rightarrow 1$: prepaginazione conveniente

Dimensione delle pagine (1)

- Le dimensioni delle pagine vanno tipicamente da 4 KB a 16 MB
  - Sono sempre potenze del 2 (2^{12} … 2^{24})
- La scelta va fatta in base a diversi fattori:
  - Dimensione della tabella delle pagine (preferibilmente piccola)
    - Più le pagine sono grandi più la tabella è piccola
    - Ogni processo deve avere una copia della tabella delle pagine
    - Memoria 4MB (2^{22})
      - Pagine 4 KB (2^{12}) ➔ tabella da 1024 pagine (2^{22} / 2^{12})
      - Pagine 16 KB (2^{14}) ➔ tabella da 256 pagine (2^{22} / 2^{14})
    - Meglio pagine grandi

Dimensione delle pagine (2)

- La scelta va fatta in base a diversi fattori:
  - Frammentazione interna
    - I processi tipicamente non riempiono in modo completo l'ultima pagina
    - In media l'ultima pagine è per metà vuota
      - Frammentazione interna di 256 Byte per pagine da 512 Byte
      - Frammentazione interna di 4 MB per pagine da 8MB
    - Meglio pagine piccole

Dimensione delle pagine (3)

- La scelta va fatta in base a diversi fattori:
  - Tempo richiesto per scrivere e leggere una pagina
    - Il tempo necessario dipende da tempo di posizionamento, latenza e tempo di trasferimento
    - Solo l'ultimo dipende dalla dimensione della pagina ed è piccolo rispetto alla somma dei primi due (8 ms + 20 ms)
    - Velocità di trasferimento 2MB/s
      - Pagina 512 Byte ➔ Tempo trasferimento 0.2 ms (1 per cento del totale)
      - Pagina 1024 Byte ➔ Tempo trasferimento 0.4 ms, totale 28.4 ms
        - Ma servirebbero 56.4 ms per 2 pagine da 512 Byte
    - Meglio pagine grandi

Dimensione delle pagine (4)

- La scelta va fatta in base a diversi fattori:
  - Copertura della località del processo
    - Pagine piccole permettono di coprire con una migliore risoluzione la località di un processo
    - Processo che occupa 200 KB di cui solo 100 KB fanno parte della località in un dato istante
      - Pagine 1 Byte ➔ Si possono trasferire solo le pagine realmente necessarie
      - Pagine 200 KB ➔ Si copia una pagina di cui 100 KB non sono usati
    - Meglio pagine piccoli, miglior utilizzo della memoria
  - Numero di assenze di pagina
    - Per lo stesso processo di prima (località da 100KB=102400 Byte)
      - Pagine 1 Byte ➔ 102400 assenze di pagina
      - Pagine 200 KB ➔ 1 assenza di pagine
    - Meglio pagine grandi

Dimensione delle pagine (5)

- La tendenza è quella di usare pagine sempre più grandi col passare del tempo

Portata della TLB

- La TLB è una memoria associativa molto efficiente che permette di memorizzare una parte della tabella della pagine
  - Molto costosa
- La sua **copertura** è data da: # elementi * dim. Pagine
  - Più la copertura è elevata più e facile caricare l'intera tabella della pagine di un processo
    - Maggiore velocità di accesso
- Aumentare la dimensione della pagine è vantaggioso
  - Ma aumenta la frammentazione interna
- Una soluzione consiste nel usare pagine a dimensione variabile
  - Grandi per grandi processi, piccole per piccoli processi
- Questo però richiede che la TLB sia gestita dal SO e non dall'architettura del calcolatore
  - È meno efficiente, ma i vantaggi dovuti all'aumento dei tassi di successo della TLB rendono questa tecnica comunque conveniente

Struttura dei programmi (1)

- In alcuni casi anche un programma scritto in modo accurato può ridurre il numero di mancanze di pagina
- Si supponga di avere pagine da 128 parole
- Il seguente codice inizializza a 0 gli elementi di una matrice da 128x128

```
int i,j;
int[128][128] data;
for ( j=0; j<128; j++)
    for ( i=0; i<128; i++)
        data[i][j] = 0;
```

- In pagine da 128 parole ogni riga della matrice occupa una pagina
- Questo codice azzera una parola per pagina e poi passa alla successiva
- Se i sistema operativo assegna meno di 128 frame a tutto il programma si hanno 128x128=16384 assenze di pagina

Struttura dei programmi (1)

- Il seguente codice invece è più efficiente

```
int i,j;
int[128][128] data;
for ( i=0; i<128; i++)
    for ( j=0; j<128; j++)
        data[i][j] = 0;
```

- In questo caso il codice azzera tutte le parola di una pagina e poi passa alla successiva
- Se i sistema operativo assegna meno di 128 frame a tutto il programma si hanno 128 assenze di pagina

- Anche le strutture dai influenzano la località
  - Le pile sono efficienti in quanto l'accesso viene sempre operato dall'alto
  - Le hash-table invece comportano per natura accessi sparsi (accessi a più pagine)
- Tuttavia le hash-table hanno un tempo di ricerca inferiore

Vincolo di I/O (1)

- L'idea è quella di bloccare alcune pagine impiegate per l'I/O in modo che non vengano sostituite
- Questo può avvenire quando l'I/O si esegue da o verso la memoria virtuale
- La seguente successione di eventi potrebbe causare questo problema:
  - Un processo emette una richiesta di I/O e viene messo nella coda del dispositivo
  - La CPU viene assegnata ad altri processi che accusano assenze di pagine
    - L'algoritmo di sostituzione sostituisce alcune pagine tra cui quella contenente l'indirizzo di I/O per l'operazione richiesta dal processo in attesa
  - Quando il processo è pronto per l'I/O e viene ricaricato in memoria
    - L'operazione di I/O parte dalla pagina specificata la quale però è ora appartenente ad un altro processo

Vincolo di I/O (2)

- Questa situazione può essere evitata introducendo il bit di vincolo nella tabella delle pagine
- Una pagine col bit di vincolo a 1 non può essere sostituita

- I bit delle pagine impegnate in I/O vengono settati ad 1 e rimessi a 0 solo al termine dell'operazione di I/O

Elaborazione in tempo reale

- La memoria virtuale garantisce un buon utilizzo del computer ottimizzando l'uso della memoria
- Tuttavia, i singoli processi possono soffrirne perché possono ricevere mancanze di pagine supplementari in esecuzione
  - Ritardi inaspettati
- Pertanto, la maggior parte dei **sistemi in tempo reale (hard real-time) ed embedded non implementano la memoria virtuale**
- **Esiste anche "una via di mezzo": ad es. Solaris**
  - Un processo può comunicare quali pagine sono per lui importanti
  - Ad utenti privilegiati è permesso il **blocco delle pagine in memoria**

Esempi di sistemi operativi

- Windows XP
- Solaris

Windows XP

- Usa la richiesta di paginazione con **clustering** che gestisce le mancanze di pagina caricando non solo le pagine su cui è avvenuta la mancanza stessa ma anche parecchie pagine dopo
- Ad ogni processo è assegnato un **working set minimo** ed un **working set massimo**
- Il working set minimo è il numero minimo di pagine garantite al processo e che sono residenti in memoria
- Ad un processo possono essere assegnate tante pagine quanto il suo working set massimo
- Quando la quantità di memoria libera cade sotto la soglia, l'**automatic working set trimming** ristabilisce il valore sopra la soglia
- Il working set trimming rimuove le pagine dai processi che hanno pagine in eccesso rispetto al loro working set minimo

Solaris

- Mantiene una lista di pagine libere da assegnare ai processi che falliscono
  - **Lotsfree**: molte pagine libere
  - **Minfree**: pochepagine libere
- **Scanrate** è il tasso di esplorazione delle pagine
  - Varia da **slowscan** a **fastscan**
- La paginazione è eseguita attraverso il processo di *pageout*
  - Esamina le pagine con un algoritmo dell'orologio modificato
- La frequenza di invocazione del pageout dipende dalla quantità di memoria libera disponibile